ASG
Software Solutions

ASG-DictionaryManager™ User's Guide

Version 2.5
Publication Number: DYR0200-25
Publication Date: November 1996

ASG Worldwide Headquarters Naples Florida USA | asg.com | info@asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 239.435.2200 Fax: 239.263.3692 Toll Free: 800.932.5536 (USA only)

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (239) 263-3692.

Company Name	**Telephone Number**	**Site ID**	**Contact name**

Product Name/Publication	**Version #**	**Publication Date**
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)
- Verify whether you received an ASG Service Pack for this product. It may include information to help you resolve questions regarding installation of this ASG product. The Service Pack instructions are in a text file on the distribution media included with the Service Pack.

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578	239.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.222	49.89.45716.400	support.de@asg.com
Singapore	65.6332.2922	65.6337.7228	support.sg@asg.com
All other countries:	1.239.435.2200		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578		
Asia	65.6332.2922	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**Netherlands**	00.800.3354.3578
Denmark	00.800.9932.5536	**New Zealand**	00.800.9932.5536
France	00.800.3354.3578	**Singapore**	001.800.3354.3578
Germany	00.800.3354.3578	**South Korea**	001.800.9932.5536
Hong Kong	001.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Ireland	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536	**Thailand**	001.800.9932.5536
Japan/IDC	0061.800.9932.5536	**United Kingdom**	00.800.9932.5536
		All other countries	1.239.435.2200

ASG Web Site

Visit http://www.asg.com, ASG's World Wide Web site.

Submit all product and documentation suggestions to ASG's product management team at http://www.asg.com/asp/emailproductsuggestions.asp.

If you do not have access to the web, FAX your suggestions to product management at (239) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication's front cover.

Contents

Preface

This *ASG-DictionaryManager User's Guide* describes ASG-DictionaryManager (herein called DictionaryManager). DictionaryManager is the corporate dictionary-driven interchange system that supports the exchange of definitions between multiple vendor dictionaries and/or directories.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "Introduction to DictionaryManager" provides an introduction to DictionaryManager.
- Chapter 2, "Translation Rules for Dictionary Members" explains translation rules and their definitions.
- Chapter 3, "Export to Another Dictionary" describes the export engine for User Selected Dictionary (selectable unit DYR-TE00), which is a prerequisite for any export selectable units.
- Chapter 4, "Import Executive Routines and Variables" describes the different executive routines and variables in DictionaryManager.
- Chapter 5, "Format Lines and Parameter Numbers" describes the format lines and parameter numbers common to all member types as well as those in basic and ASG-DesignManager member types.
- Chapter 6, "Syntax of Commands and Member Types" explains the syntax used in all DictionaryManager commands and member types.

Publication Conventions

Allen Systems Group, Inc. uses these conventions in technical publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.

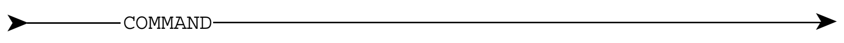
These conventions apply to syntax diagrams that appear in this publication. Diagrams are read from left to right along a continuous line (the "main path"). Keywords and variables appear on, above, or below the main path.

Convention	Represents
➤➤	at the beginning of a line indicates the start of a statement.
➤◄	at the end of a line indicates the end of a statement.
———➤	at the end of a line indicates that the statement continues on the line below.
➤———	at the beginning of a line indicates that the statement continues from the line above.

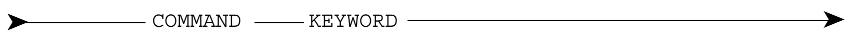
Keywords are in upper-case characters. Keywords and any required punctuation characters or symbols are highlighted. Permitted truncations are not indicated.

Variables are in lower-case characters.

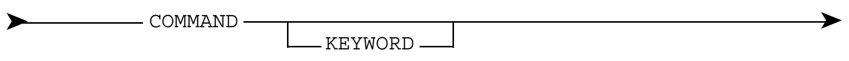
Statement identifiers appear on the main path of the diagram:



A required keyword appears on the main path:

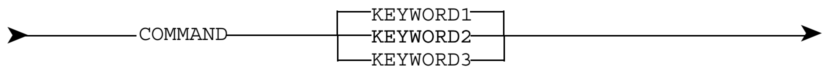


An optional keyword appears below the main path:

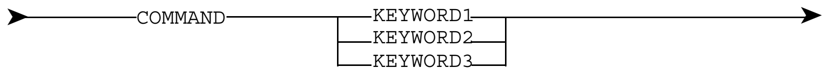


Convention	Represents

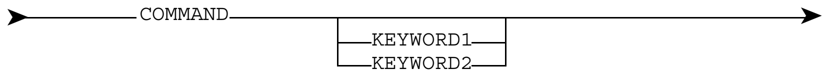
Where there is a choice of required keywords, the keywords appear in a vertical list; one of them is on the main path:



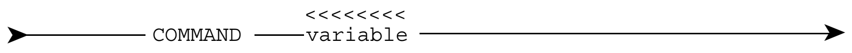
or



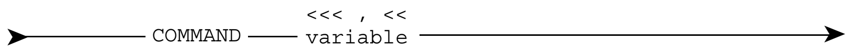
Where there is a choice of optional keywords, the keywords appear in a vertical list, below the main path:



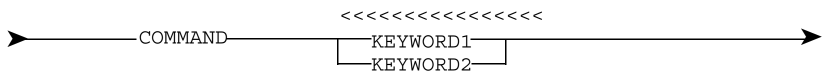
The repeat symbol, <<<<<<, above a keyword or variable, or above a whole clause, indicates that the keyword, variable, or clause may be specified more than once:



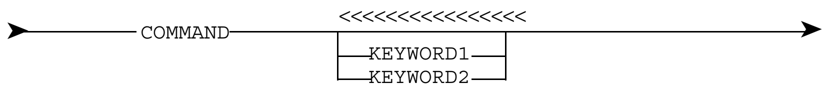
A repeat symbol broken by a comma indicates that if the keyword, variable, or clause is specified more than once, a comma must separate each instance of the keyword, variable, or clause:



The repeat symbol above a list of keywords (one of which appears on the main path) indicates that any one or more of the keywords may be specified; at least one must be specified:



The repeat symbol above a list of keywords (all of which are below the main path) indicates that any one or more of the keywords maybe specified, but they are all optional:



1 Introduction to DictionaryManager

DictionaryManager enables you to transfer information between different ASG-Manager Products (herein called Manager Products) dictionaries and between Manager Products dictionaries and those of other vendors. It consists of a nucleus, an export engine, generic import functions, and additional selectable units. The selectable unit User Defined Output allows you to define the content and layout of dictionary reports (see the publication *ASG-Manager Products User Defined Output*).

DictionaryManager provides the means for transferring information between your corporate dictionary and other dictionaries. The DictionaryManager nucleus (selectable unit DYR-DY01) is an environmental, together with the ASG-ControlManager (herein called ControlManager) prerequisite nucleus, for other Manager Products.

Exporting Information

With the DictionaryManager nucleus and the corporate dictionary definition export engine for the User Selected Dictionary (selectable unit DYR-TE00), you can form translation rules and translate definitions of Manager Products dictionary member types to the format of a dictionary system you select. You can then transfer the translated definitions to an external file.

Other selectable units are specifically tailored to particular dictionary, such as Cullinet's IDD. With the corporate dictionary definition export for IDD (selectable unit DYR-TE08), you can export to IDD and IDMS using ASG-supplied translation rules.

The fundamental principle of DictionaryManager is to define the rules by which members of a particular type (such as ITEM) are translated from one dictionary's format to another.

The rules are held as TRANSLATION-RULE members, so that once set up they can always be used to translate members of the specified type to the specified target dictionary. ASG provides a preferred set of TRANSLATION-RULES as part of each selectable unit related to a specific non-ASG dictionary.

A TRANSLATE command uses the TRANSLATION-RULEs to convert members to source input statements for the target dictionary and hold them in a USER-MEMBER on the MP-AID.

Another command TRANSFERs the translated members to an external file to be used as source input statements to the target dictionary.

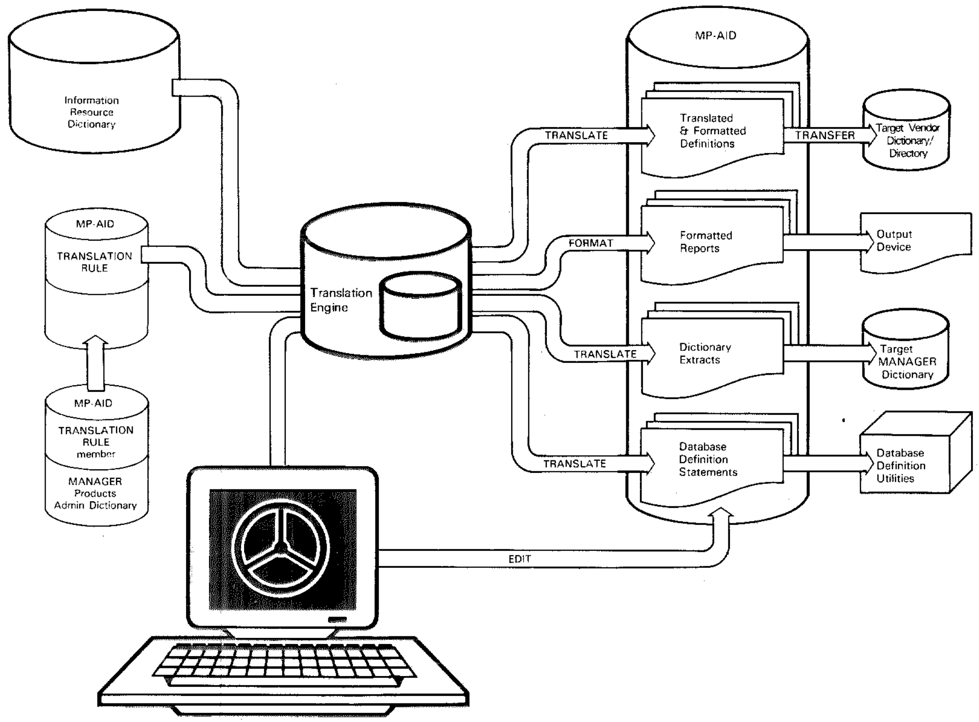

Information Resource Dictionary
MP-AID
TRANSLATION RULE
MP-AID
TRANSLATION RULE member
MANAGER Products Admin Dictionary
Translation Engine
MP-AID
TRANSLATE
Translated & Formatted Definitions
TRANSFER
Target Vendor Dictionary/ Directory
FORMAT
Formatted Reports
Output Device
TRANSLATE
Dictionary Extracts
Target MANAGER Dictionary
TRANSLATE
Database Definition Statements
Database Definition Utilities
EDIT

Figure 1. DictionaryManager: Export

Before issuing the first TRANSLATE command in any DictionaryManager session, you may want to SET the system to select the default TRANSLATION-RULES (either those supplied by ASG or ones created by you).

Summary

DictionaryManager exportation is structured to be used in this way:

- Define your TRANSLATION-RULEs (unless you are using those provided by ASG). Define them as members on the Manager Products Administration dictionary, and then construct them to the MP-AID. You need to redefine them only if the rules change.
- Alternatively, amend (and reconstruct) TRANSLATION-RULEs as required. For example, you might substitute form-description in VARIABLE clauses and replace parameter-numbers in USER-EXIT clauses.
- At the beginning of each session in which you will transfer information between dictionaries, use the SET TRANSLATION command to specify the translation details to be used, by default, by subsequent TRANSLATE commands. You can query these settings using the QUERY TRANSLATION command. The SET TRANSLATION command can be used at any time during the session to change the defaults.
- TRANSLATE members of specific types, as required. The translated source is held on the MP-AID.

- TRANSFER the translated members to an external file, ready for input to the target dictionary.
- If necessary, prepare your target dictionary to receive member definitions translated from a Manager Products dictionary, using the target dictionary's extensibility feature.

Importing Information

Using *generic import functions*, you can import objects from any system into a DictionaryManager repository. These functions are provided by the EXTRACT, RECONCILE, PREVIEW, and POPULATE commands.

You can also use predefined import functions to import information from COBOL and PL/I data descriptions and from these environments:

- DB2
- SQL/DS
- ADW
- IEW

You can use generic import functions to document any system in your repository. Large program libraries and database systems such as IMS can be extracted and translated into repository definitions.

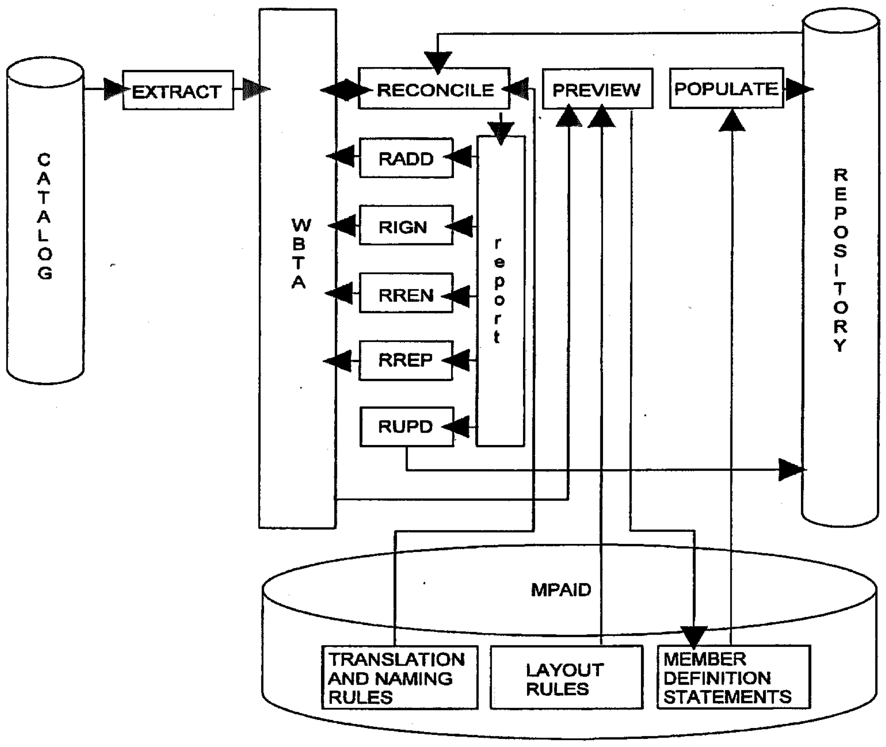


Figure 2. DictionaryManager: Import

The main benefits of documenting your systems in the repository are obtained when you use the repository to automatically generate new systems or new parts of existing systems to design a new or modified database.

These are the integral parts of generic import functions:

- An import procedure to import objects from an external environment into the repository. The import procedure consists of four stages: extract, reconcile, preview, and populate.
- The Manager Products Procedures Language which allows you to combine Manager Products commands with directives that provide standard programming capabilities (including conditional logic).
- A WorkBench Translation Area (WBTA) in virtual storage on which information can be stored (in named Procedures Language variables) and then manipulated using the Procedures Language to suit your purposes.

Refer to "The Import Procedure" on page 5 for details of the import procedure.

Refer to the *ASG-Manager Products Dictionary/Repository User’s Guide* for details of the benefits of using a repository.

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for details of the EXTRACT, RECONCILE, PREVIEW, and POPULATE commands.

Refer to the publication *ASG-Manager Products Procedures Language* for details of the Procedures Language.

Refer to these publications for details of the predefined import functions:

- *ASG-Manager Products Relational Technology Support: DB2*
- *ASG-Manager Products Relational Technology Support: SQL/DS*
- *ASG-Manager Products Tools Support: Integration with ADW/IEW*
- *ASG-DataManager Automation of Set Up*

Note:

You need a working knowledge of Manager Products' Procedures Language in order to develop generic import functions effectively.

Data Analysis

Before you import external information into a repository, you need to consider what structure the information will have in the repository. For example, if you are importing a COBOL program, you may decide that the program will be represented in your repository by a member type called PROGRAM and that it will have CALLS attributes documenting the external modules it calls.

So, you must decide what you want to store in the repository and what form and structure it will have in the repository.

How you decide to represent the information in your repository will, in turn, depend on what use you intend to make of the information. For example, to keep track of which modules a program calls, you can have CALLS attributes in a PROGRAM member (as described above). If you are more concerned with which files a program updates, you can extract different information and represent it appropriately in the repository.

Once you have decided what you want to import to a repository, you must plan to organize the extracted information on the WBTA in a way that supports the (subsequent) reconcile and preview stages of the import procedure.

The Import Procedure

Introduction

You can use generic import functions to import objects from an environment that is external to Manager Products with the four-stage import procedure:

Extract. Read an external file or dataset into the WBTA using the EXTRACT command.

Reconcile. Generate proposed member names and member types from the extracted information, compare them with the current contents of the Manager Products repository, and make adjustments as necessary using the RECONCILE command.

Preview. Generate proposed repository definitions and view them using the PREVIEW IMPORT command.

Populate. Populate a repository with the definitions on the WBTA using the POPULATE command.

Extract, Reconcile, and Preview Executive Routines

Each of the first three stages of the import procedure (extract, reconcile, and preview) must be supported by either corporate or user executive routines that you write. These are invoked with the USING keyword in the EXTRACT, RECONCILE, and PREVIEW commands respectively.

Extract executive routines control the way in which extracted information is stored on the WBTA. The extracted information must be stored in a way that enables the subsequent stages to access the information they require.

Reconcile executive routines control the way in which information about the extracted objects is translated into proposed repository member name and member types. The translated information is displayed in the reconciliation report (produced by the RECONCILE command).

Preview executive routines control the translation of the extracted information into repository definition statements which may subsequently be added to a repository.

For generic import functions, you should write the routines for a particular application of the import procedure. For the predefined import functions, ASG provides the routines (some of which are tailorable). ASG also provides you with extensive examples of such executive routines.

Refer to Appendix A, "Importing a COBOL Program," on page 103 and Appendix B, "Importing an Entity-relational Model," on page 131 for examples of generic import.

2 Translation Rules for Dictionary Members

Introduction to Translation Rules

This chapter explains translation rules and their definitions. The capability to define translation rules is provided by DictionaryManager's nucleus (DYR-DY01).

Chapter 3, "Export to Another Dictionary," on page 33 explains how to apply translation rules to members of a Manager Products dictionary and how to export the translated definitions to another dictionary system.

DictionaryManager enables you to translate Manager Products dictionary member types such as ITEM, GROUP, and FILE (and any UDS synonyms) to the format of equivalent member types in other dictionary systems. This is done in two stages:

- Define TRANSLATION-RULEs for translating each member type from the Manager Products dictionary to another dictionary
- TRANSLATE members of the Manager Products dictionary, using the rules.

The TRANSLATION-RULEs are held on the MP-AID. You may have a number of different rules, including the set provided by ASG. A SET TRANSLATION command is used to establish the default rules to be applied for subsequent TRANSLATE sessions.

How to Define TRANSLATION-RULEs

The process of defining TRANSLATION-RULEs is based on a selection of clauses and attributes. The process uses a numbering system and several formatting procedures.

Forming TRANSLATIONRULEs

This section explains the system of selection of clauses and parameters. The procedure used depends on a numbering system and is implemented differently depending on whether the data to be translated is in a clause that is repeated or not. Formatting is also explained.

The TRANSLATION-RULE member contains the rules for translating dictionary member definition syntax between a Manager Products dictionary and another target dictionary.

The members are developed in the Manager Products Administration dictionary and transferred to the MP-AID by the Systems Administrator. The rules are then available for use in translating members from any Manager Products dictionary. (Once it is on the MP-AID, a TRANSLATION-RULE member can be referred to as a TRUL.)

This is the basic syntax of TRANSLATION-RULE members:

```
TRANSLATION-RULE FOR MANAGER-DICTIONARY member-type
                        MP-AID-NAME mp-name

VARIABLE Vm  ⎧ IS clause                  ⎫
             ⎨ USER-EXIT i PASSING arg    ⎬
             ⎩ translate clause           ⎭

CONTENTS

⎧ format-line  ⎫
⎩ comment-line ⎭ ...
```

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for the complete syntax.

The method for defining TRANSLATION-RULEs is based on a clause and attribute selection. It considers a member definition to be a collection of clauses of information.

There is a numbering system that identifies each clause or attribute of syntax (as it would be REPORTed using the dictionary management REPORT command), and another for identifying items of information within each clause. A clause is identified by a format line number. An item of information within a clause is identified by a parameter number.

However, other dictionary systems will differ from a Manager Products dictionary in the sequence and content of their clauses of syntax. DictionaryManager overcomes this by:

- Allowing the clauses to be output in any order
- Using special techniques to translate repeating clauses
- Introducing VARIABLES to output parameters only under specific circumstances

If your target dictionary requires further specialized manipulation of these parameters, you can customize your variables through a TRANSLATE user exit.

For member types which refer to other members with a reference such as a CONTAINS clause (for example GROUP), a TRANSLATION-RULE can include a clause to TRANSLATE the contained members. This is so that when the TRANSLATION-RULE is processed by a TRANSLATE command, the contained members are automatically translated as well.

Formatting characters allow you to control the positions of the translated items of information within a line when it is output.

Character strings can be included in format lines.

Refer to "Formatting Characters and Character Strings" on page 11 for details on formatting characters and character strings.

The Numbering System Used in TRANSLATION-RULEs

The numbering system is based on the format of a Manager Products dictionary report. Consider the report for a GROUP named ADDRESS, defined in a Manager Products dictionary as:

```
GROUP
ALIAS  'DELIV-ADDR',  'INVCE-ADDR',  'STMNT-ADDR' ,COBOL
'ORDER-ADDRESS'
DESCRIPTION  'CUSTOMER ADDRESS  IN ORDER PROCESSING SYSTEM'
CATALOG  'SALES',  'ACCOUNTS'
HELD-AS
CONTAINS NAME
       ,ADDRESS1
       ,ADDRESS2
       ,ADDRESS3
       ,CODE
;
```

The standard format of the report is:

```
REPORT OF GROUP ADDRESS

EDITION 2 ENCODED BY JEH AT 11.31.30 ON 15 JAN 1998
STATUS MCS
PROTECTION: NONE

ALIAS
       COBOL ORDER-ADDRESS
       DELIV-ADDR
       INVCE-ADDR
       STMNT-ADDR

CATALOGUED AS
       SALES
       ACCOUNTS

DESCRIPTION
       CUSTOMER ADDRESS IN ORDER PROCESSING SYSTEM

THIS MEMBER IS DIRECTLY REFERRED TO 3 TIMES

THIS MEMBER CONTAINS 5 DIRECT REFERENCES

GROUP ADDRESS

FORM HELD-AS

CONTAINS
       ITEM NAME
       ITEM ADDRESS1
       ITEM ADDRESS2
       ITEM ADDRESS3
       ITEM CODE
```

Each clause of the report is in a fixed position, and in a fixed format. For example:

```
DESCRIPTION 'CUSTOMER ADDRESS IN ORDER PROCESSING SYSTEM'
```

always appears after the ALIAS and CATALOG information, beginning with the keyword DESCRIPTION, as above. This clause has been allocated a format line number (FM08:), and the two items within it were allocated the parameter numbers:

D46 (DESCRIPTION)
D47 (CUSTOMER ADDRESS IN ORDER PROCESSING).

The output specified by each format line normally begins on a new physical line and in column one. You can insert new lines, spaces, and fixed character strings by means of one or more formatting characters.

Refer to "Format Lines and Parameter Numbers Common to All Member Types" on page 53 for a list of the format lines and parameter numbers.

Formatting Characters and Character Strings

Formatting characters allow you to control the positioning of the translated items of information within an output line.

If you use the FM format lines, parameter numbers, and variables as described, your translated output will begin on a new physical line, in column one.

If the members of your target dictionary need to be defined with each line beginning in a column other than one, or have special spacing requirements, you can specify these requirements in your TRANSLATION-RULE by inserting certain characters, designated as specifiers.

The specifiers, including the parameter numbers, must be separated from each other by commas. For example:

```
* ALIAS CLAUSE FORMAT LINES, TO ILLUSTRATE FORMATTING
* CHARACTERS
FM02: C5, D42, X, D43
FM03: C5, D42
- D43.10.R
```

You can also insert character strings, to build up a complete member definition in the format required by the target dictionary.

If you want to fit values into specified field widths where the specified field width is not the same as that of the value to be translated, special procedures are available.

List of Formatting Characters and Their Functions

These are the formatting characters and their functions.

Character	Description
*	An asterisk begins a comment line, which does not appear in the output. Each comment line must be contained in one input line, but you can use any number of consecutive comment lines.
C*nn*	The output that follows begins in column *nn*.
*n*X	Causes *n* spaces to be output.
.*mm*	A number (*mm*) attached to a parameter number defines a field-width of *mm* character spaces for the value output from the parameter number (D*n*, S*n*, V*n*, or U*n*). The period separates the field-width specification from the parameter number. There must be no spaces between the field-width, its preceding period, and the parameter number. If the field contains characters, the output is left justified in the field. If the field contains integers, they are right justified in the field. Leading or trailing spaces are included, as described in the note below.

Character	Description
.R	Immediately follows a parameter number or a field-width specification to right justify the output from the preceding format specifier in the field (D43. l0.R). The period separates R from the field-width specification with no intervening spaces, as for *mm*.
q(*s*1,*s*2,...)	The output from the specifiers *s*1,*s*2,... is repeated *q* times [for example 2(X,D43)]. If only one specifier is given, the brackets are optional; for example 4X.
.C	Immediately follows a field-width specification to center the translated output from the preceding parameter number. The period (full stop) separates C from the field-width specification (with no intervening spaces, as for .*mm*).
/	A slash character forces a new line in the output.

These syntax conventions are used in structuring lines.

Character	Description
-	Whenever a format line carries on to a new line of input the new line must begin with a hyphen.
,	A comma must be included between specifiers, including format parameters, but not at the beginning or end of a line.
	Spaces may be freely interspersed in the output line specification without effect on the output.

Character Strings

You can include character strings in format lines, either on a separate format line, or interspersed with parameter numbers, variables, and other formatting characters. Each character string must be enclosed in a pair of single quote marks. You can include any printable character, as well as spaces, within a character string.

If you want to include a single quotation mark as a printable character, then you must include a pair of quote marks in the text at a suitable point. For example if format line FM04 were:

```
FM04: 'END OF ''SYNONYM''  NAMES'
```

then the corresponding output would be:

```
END OF "SYNONYM" NAMES
```

Example

The CONTENTS clause of a TRANSLATION-RULE member for translating a GROUP might be:

```
* ADD GROUP ELEMENT USING MANAGER PRODUCTS MEMBER NAME
FM01:C7,  'ADD ELEMENT ',  D23
* USE FIRST THREE MANAGER DICTIONARY ALIASES TO DEFINE
* TARGET DICTIONARY SYNONYMS
FM02: C7, 'ELEMENT NAME SYNONYM IS "', D43,'"'
FMO3R2: C7, 'ELEMENT NAME SYNONYM IS "', D43,'"'
* ADD SUBORDINATE ELEMENTS FROM GROUP CONTAINS LIST
FM01: C7, 'SUBORDINATE ELEMENTS ARE'
FM192:
FM192L: C12, D317
- /, Cl7, 'SUB-ELEMENT SYNONYM IS "' , D321,'"'
FM193:  C7, '-'
FM9999:
```

This would cause the member for the GROUP ADDRESS, given in the example in the General Details panel, to be translated as:

```
ADD ELEMENT ADDRESS
ELEMENT NAME SYNONYM IS "DELIV-ADDR"
ELEMENT NAME SYNONYM IS "INVCE-ADDR"
ELEMENT NAME SYNONYM IS "STMNT-ADDR"
SUBORDINATE ELEMENTS ARE
     NAME
     ADDRESS1
     ADDRESS2
     ADDRESS3
     CODE
            SUB-ELEMENT SYNONYM IS 1 "POSTCODE"
```

Fitting Values into Specified Field-widths

When a TRANSLATION-RULE is applied, if a field-width (*mm*) is specified and a value to be translated in the field is longer or shorter (length N) than that specified by *mm*, then the following rules apply. Otherwise, (that is, if *mm*=N), the parameter value fills the field exactly.

For parameters whose values are character strings:

- Where no justification is specified:
  - — If *mm* is greater than N, the parameter value is left justified in the specified field, filled with trailing spaces.
  - — If *mm* is less than N, the parameter value is truncated from the right to fill the field exactly.
- Where right justification is specified (in a type D*m*/U*m*/V*m.mm*.R Parameter Number):
  - — If *mm* is greater than N, the parameter value is right justified in the specified field, filled with leading spaces.
  - — If *mm* is less than N, the parameter value is truncated from the left to fill the field exactly.
- Where centering is specified (in a type D*m*/U*m*/V*m.mm*.C Parameter Number):
  - — If *mm* is greater than N, the parameter value is centered in the specified field, filled with leading and trailing spaces.
  - — If *mm* is less than N, the parameter value is centered in the field, truncated at both end to fill the field exactly.

For parameters that have integer values:

- Right justification is assumed:
  - — If *mm* is greater than N, the parameter value is right justified in the specified field, filled with leading spaces.
  - — If *mm* is less than N, the parameter value is filled with asterisks (*) to denote overflow.

Format Lines and Parameter Numbers

Format Lines

The CONTENTS clause of a TRANSLATION-RULE member contains a series of format line statements. A format line enables you to identify items of information in a member definition of one dictionary, and reorganize them into the format for definition as a member of another dictionary. It has the form:

```
format-line-number   output-line-specification
```

Format line numbers are introduced below. The output-line specification includes parameter numbers, variables, formatting characters, and character strings.

This is an example of a simple format line:

```
FM08: C7, D46, 5X, D47
```

where `FM08:` is a format line number, `D46` and `D47` are parameter numbers, and `C7` and `5X` are formatting characters.

The format line number specifies a new clause for output, and identifies the Manager Products classification of the information it contains. It has the form:

`FM`*`n`*`:`

where `FM` identifies a new format line and *`n`* identifies the clause or data to be transferred. It must be in the range 1 through 9999. The colon (`:`) delimits the end of the format line number.

For example, format line FM02: contains ALIAS information. Format line numbers FM01: and FM9999: are reserved as special indicators. FM01: outputs system parameters, and FM9999 indicates the end of the TRANSLATION-RULE.

A format line can occupy more than one line in the TRANSLATION-RULE definition. Whenever a format line carries on to a new line, the new line must begin with a hyphen (-). After translation, the maximum total length of a line which can be generated by a format line is 132 characters. A longer generated line will be truncated to 132 characters.

The translated clause can also occupy more than one line. A slash (/) inserted in the format line will force a new line in your translated output.

The information on a format line or continuation format line will be translated if it contains:

- Text only
- Parameter numbers or variables, of which at least one has a value that is not NULL

If it contains only NULL valued parameter numbers or variables, output of any text fields (such as ALIAS IS) in that clause is suppressed.

An `FN`*`n`*: format line indicates a skip to a new line. Use the form `FN`*`n`*`:` to cause the parameters to be treated as part of the preceding `FN`*`n`*`:` line (without skipping to a new line).

These restrictions apply to the use of FN lines:

- The first format line in the CONTENTS clause of a TRANSLATION-RULE cannot be an FN line.
- An FN line cannot be specified for a repeating format line; multiple values of the same parameter may not appear on the same physical output line. In terms of the groups of three format numbers, FN*n*+1:... is invalid. A specification which refers to only one occurrence of of repeating information is valid (FN*n*:... and FN*n*+1L:... are valid).
- An FN line cannot follow a processing loop. In terms of the groups of three format numbers, an FN... line may not immediately follow an FN*n*+2:... line or an FM*n*+2 line.

Parameter Numbers and Format Lines

In order to identify the various components of a given dictionary member, each standard attribute has been assigned a parameter number. Parameter numbers have the form D*n* or U*n* , where *n* is an integer which identifies the parameter. For example, the ALIAS attribute is given the parameter D43.

The D parameters represent standard attributes and clauses of member definitions, while U parameters represent user-defined attributes (defined by the User Defined Syntax facility, selectable unit CMR-UD1). The format line and parameter numbers for user-defined attributes are documented in the hierarchy member from which the UDS table was generated.

The format line number selects the clause from which information is to be extracted, while the parameter numbers select the data and keywords from the clause. As a result, most parameter numbers are associated with a particular format line.

For example, the format lines and parameter numbers are allocated to a sample ALIAS definition as follows:

```
FM02: D41 D42 D43.........
   ALIAS    COBOL     ORDER-ADDRESS
```

However, parameter numbers D1 through D40 may be used on any format line. For example, the parameter number D23, which gives the name of the member being translated, could be placed on format line FM01:, FM08:, or any other line.

If global or local variables are set using User Defined Commands (selectable unit CMR-UD05), then certain of these variables are available for use. Parameter numbers D8 through D20 are reserved for use as global and local variables. D8 through D17 correspond to global variables G0 through G9. D18, D19, and D20 correspond to local variables L0, L1, and L2.

Refer to "Format Lines and Parameter Numbers Common to All Member Types" on page 53 for the list of format lines and parameter numbers, and to the publication *ASG-Manager Products User Defined Syntax*.

Format Line Grouping

Format lines are generally grouped in threes. For example, format line numbers FM02:, FM03:, and FM04: all output the ALIAS clause.

The three format lines have these purposes:

- The first line (FM*n*) translates the first occurrence of the data contained in the clause. For example, a format line number FM02: would translate the first ALIAS entry, such as:

  ```
  ALIAS  'DELIV-ADDR'
  ```

- The second line (FM*n*+1) controls translation of the subsequent occurrences of the data in the clause. For example, format line number FM03: would translate any additional ALIAS entries, such as:

  ```
  ,'INVCE-ADDR', 'STMNT-ADDR'
  ```

- The third line (FM*n*+2) allows a terminating line to be included at the end of the translated clause. It contains no data from the clause, but it may contain parameters which are generally available or trailing information.

For example, consider a GROUP member named ADDRESS which contains:

```
ALIAS  'DELIV-ADDR', 'INVCE-', 'STMNT-ADDR'
```

If the format lines below are included in the TRANSLATION-RULE member, then all the ALIASes would be translated, followed by the semi-colon as a terminator.

```
FMO2: (other specifiers)
FM03: (other specifiers)
FM04: ':'
```

The format line groups FM59:, FM60: and FM61:, and FM19l:, FM192:, and FM 193: are special cases. FM59: and FM19l: contain heading information and general parameters only and do not refer to the first occurrence of data in the clause referenced by the format line. All such data is controlled by format lines FM60: and FM192:.

Translating Data from a Clause that Does Not Repeat

Clauses are treated according to whether the data corresponding to the format lines may repeat or not.

Format lines may be specified in any order. They will be processed in the order in which they are encountered. You can use them in their group of three, or independently of it. You can also follow a format line from one group with those from other groups, and repeat format lines.

For example, if the target dictionary format requires a member definition to begin with one line of DESCRIPTION, followed by all the ALIASes and no terminator, you could use the sequence:

```
FM08:
FM02:
FM03:
```

For simple, non-repeating information, only the first format line should be used.

Translating Data from a Clause that Repeats

Introduction

Format lines may be specified in any order. They will be processed in the order in which they are encountered. You can use them in their group of three, or independently of it. You can also follow a format line from one group with those from other groups, and repeat format lines.

For example, if the target dictionary format requires a member definition to begin with one line of DESCRIPTION, followed by all the ALIASes and no terminator, you could use the sequence:

```
FM08:
FM02:
FM03:
```

When information is repeated in a main clause, such as in the ALIAS example, the first format line specifies the first occurrence of the repeating information, and the second format line specifies all subsequent occurrences.

However, some clauses are subsidiary to a main clause. For example, if the CONTAINS clause of a GROUP member held these repeating lines of an address, the IF clause is a subsidiary clause:

```
,ADDRESS1
,ADDRESS2
,ADDRESS3
,POSTCODE ELSE ZIP IF ADDRESS3 EQ 'USA'
```

To ensure that all occurrences of a subsidiary clause are processed, it is necessary to set up a procedure to repeat the translation on each occurrence of the subsidiary clause. You can do this with a processing loop.

Controlling Translation of Items of Data after the First Occurrence

Consider a GROUP member named ADDRESS that contains:

```
DESCRIPTION 'DESCRIPTION LINE ONE'
            'DESCRIPTION LINE TWO'
            'DESCRIPTION LINE THREE'
```

These format lines translate all the DESCRIPTION lines:

```
FM08 : (other specifiers)
FM09 : (other specifiers)
FM010: ';'
```

In this example, you could translate only the DESCRIPTION LINE ONE and the terminator by omitting the second line:

```
FM09 : (other specifiers)
```

Alternatively, you could translate only up to DESCRIPTION LINE TWO by adding a repeat count, R*i* , to the format line number of the second line in the group (FM09R1). This ensures that only one DESCRIPTION LINE after the first is translated.

The format of the Repeat Count is R*r* , where *r* is the number of occurrences in the clause or data that are to be translated.

If a format line number FM03R3: is used, the lines output would include the second, third, and fourth DESCRIPTION entries, assuming they were specified in the definition of a dictionary member. Fifth or subsequent DESCRIPTION entries would be omitted.

There are other special procedures available for translating clauses.

Processing Loops

Processing loops provide a mechanism for handling repeating information. They must be used when information from other format lines is to be interspersed with information from a repeating clause.

You can repeat a process for each occurrence of the contained member within a GROUP by means of this processing loop:

```
FN192:
FM192L: 'ADD ELEMENT ', D317
-     'ELEMENT NAME SYNONYM IS ', D321
FM193:
```

If you applied that processing loop to a Manager Products GROUP member containing the items:

```
NAME
ADDR
PHON KNOWN-AS TEL
```

this would be the output:

```
ADD ELEMENT NAME
ADD ELEMENT ADDR
ADD ELEMENT PHON ELEMENT NAME SYNONYM IS TEL
```

The beginning of a processing loop is identified by an FMR+1 format line with no specifiers (FM192:). The format line identified by FM192L: (FM*n*+1L) contains the specification of what must be translated for each occurrence of the repeating information. The end of the processing loop is indicated by a normal FM*n*+2 format line.

In the example above, for each occurrence of a member in the GROUP's CONTAINS clause (that is each occurrence of FM192:), a new ADD ELEMENT statement is generated. If the member has a corresponding KNOWN-AS clause, a SYNONYM clause is generated within the ELEMENT definition.

FM193: identifies the end of the processing loop and control will return to the immediately preceding FM192: (no L) until all occurrences of FM192: are processed. Any output specification included in line FM193: is processed after all occurrences of FM192:.

Interspersing Information from Other Format Lines with Information from a Repeating Clause

In the example below the processing loop will output an ADD ELEMENT statement for each form/version of an ITEM, each with the first four ALIASes of the member.

```
FM60:
FM60L: 'ADD ELEMENT', X, D23, X, 'VERSION', X, D112
FM02: 'SYNONYMS ARE', X, D43
FM03R3: C3, D43
FM61: '-'
```

Processing loops may be nested. An inner loop will repeat for each occurrence of repeating information which occurs within a single occurrence of the section defined by the outer loop. For example:

```
FM60:
FM60L: 'ADD ELEMENT', X, D23, X, 'VERSION', X, D112
FM465:
FM465L: D157, X, 'MAY BE', X, D158
FM466:
FM61: '-'
```

This will define a new ELEMENT on the target dictionary for each form/version, with the FORMAT specifications of the subsidiary CONTENTS and ELSE clauses that are within the current occurrence of the form/version.

Note:

It is recommended that you only use nesting of processing loops to correspond to subsidiary clauses which naturally repeat within the main format lines.

Refer to "Translating Data from a Clause that Repeats" on page 17 for a description of translating a clause that repeats.

Concatenating Clauses from Different Format Lines

The target dictionary may require a line of a member definition to be made up of part of one line of an ASG member definition, and part of another line (of the same ASG member definition).

For example, you might want an output line containing the PICTURE clause from an ITEM member definition, followed by the RANGE clause. The PICTURE clause is specified by: FM59: D133, D134, and the RANGE clause is specified by FM467: D153, D609, D610, D611.

If you specify a format line beginning FN after an FM format line, the output line will begin with the parameters specified in the FM format line. The parameters specified in the FN format line will be concatenated onto the end of the output line. For example:

```
FM59: C5, D133, X, D134
FN467: X, D153, X, D609, X, D610, X, D611
```

will cause the output of:

```
PICTURE 'picture-string' RANGE lower-limit TO upper-limit
```

You can specify more than one FN format line following an FM format line.

Using Variables

How Variables are Used

Variables allow you to vary the allocation and/or content of items of information within format lines. For example, members of the Manager Products type GROUP may have a FORM of ENTERED-AS, HELD-AS, REPORTED-AS, and/or DEFAULTED-AS. The parameter number for FORM is D301.

Suppose you are translating to a target dictionary whose equivalent terms for these are INPUT, FILED, OUTPUT, and DEFAULT respectively.

You can ensure that these variations are translated correctly by placing the following statement in your TRANSLATION-RULE, before the CONTENTS section:

```
VARIABLE V1 IS 'INPUT' IF D301 EQ 'ENTERED-AS'
      ELSE 'FILED' IF D301 EQ 'HELD-AS'
      ELSE 'OUTPUT' IF D301 EQ 'REPORTED-AS'
      ELSE 'DEFAULT' IF D301 EQ 'DEFAULTED-AS'
```

and then, in your CONTENTS section, a format line for the Manager Products dictionary FORM clause:

```
FM185: V1
```

If there is no match, the variable is set to null.

A variable is defined only in terms of the current TRANSLATION-RULE. Its value, once set, remains unchanged until it is reset by reprocessing the format line containing that variable, or processing another format line containing that variable.

Referenced members can be translated automatically using variables.

Example of Using Variables

A target dictionary may not have facilities to cope with the KNOWN-AS clause. It may be necessary to put commands in a TRANSLATION-RULE member for a GROUP to define the CONTAINed members KNOWN-AS names, if specified, as an alias or synonym on the target dictionary.

In this case, the variable definition might be this:

```
VARIABLE V2 IS D317 IF D321 NE NULL
VARIABLE V3 IS D23 IF D321  NE NULL
```

where parameter numbers are assigned like this:

D23	Member name in any format line (the GROUP name here)

D317	Member name in the CONTAINS clause
D318	Version number in the CONTAINS clause
D318	Version number in the CONTAINS clause

Commands to define members on the target dictionary might be put in format lines in the CONTENTS section, to operate on the variables:

```
FM192: 'MODIFY ELEMENT ', V2
- /,'INCLUDE ELEMENT NAME SYNONYM ',D321,'FOR GROUP ',V3, '.'
```

Special Processing with the User Exit

The user exit provides a mechanism for special processing not handled by the TRANSLATION-RULE processing. For example, you can multiply two parameters together.

The user exit processing is invoked by declaring a variable clause with this format:

```
VARIABLE Vm USER-EXIT user-exit-no (PASSING parameter-numbers)
```

When V*m* is encountered within a format line, the user exit module MPDX1 is called, passing the user exit number and parameters. The user exit module returns the value of V*m*.

A maximum of fifteen parameters can be passed to the user exit module along with the user exit number. This information is passed in the format of an IDCB control block.

The user exit processing can set a return code to indicate either that the TRANSLATION-RULE processing should continue upon return from the user exit and a warning message should be output before continuing, or that an error message should be output and the translation terminated.

The ASG-supplied code for MPDX1 contains coding specific to the translation of a Manager Products ITEM PICTURE into an IDD picture. You can amend the source of the module to either add your own functions or to modify the supplied code to your own needs.

Defining Your Own Functions for the User Exit

Field Contents	**ASG Name**	**Decimal Offset**	**Form Description**	**Length Bytes**	**Values**	**Remarks (below)**
Address of user work core (60 bytes)	IDCBWKAD	0	A	4		1
Calling modules register 13	IDCBR13	4	F	4		2
Number of passed parameters	IDCBPMNO	8	H	2		3
Length of work area	IDCBWKLN	10	H	2	X'3C'	4
Function code from calling module	IDCBFUNC	12	C	2	C'00' through C'99'	5

Defining Your Own Functions for the User Exit

Field Contents	ASG Name	Decimal Offset	Form Description	Length Bytes	Values	Remarks (below)
Return code to calling module	IDCBRETN	13	X	I	X'0' X'4' X'8'	6
Address answer IDX	IDCBANSW	16	F	4		7
Address of parameter 1	IDCBPR01	20	F	4		8
Address of parameter 2	IDCBPR02	24	F	4		8
Address of parameter 3	IDCBPR03	28	F	4		8
Address of parameter 4	IDCBPR04	32	F	4		8
Address of parameter 5	IDCBPR05	36	F	4		8
Address of parameter 6	IDCBPR06	40	F	4		8
Address of parameter 7	IDCBPR07	44	F	4		8
Address of parameter 8	IDCBPR08	48	F	4		8
Address of parameter 9	IDCBPR09	52	F	4		8
Address of parameter 10	IDCBPR10	56	F	4		8
Address of parameter 11	IDCBPR11	60	F	4		8
Address of parameter 12	IDCBPR12	64	F	4		8
Address of parameter 13	IDCBPR13	68	F	4		8
Address of parameter 14	IDCBPR14	72	F	4		8
Address of parameter 15	IDCBPR15	76	F	4		8

Automatic Translation of Referenced Members

If you are defining a TRANSLATION-RULE for a member type such as GROUP which references other members, you can arrange for them to be translated automatically when the member type you are defining is translated. In this way, you do not have to translate each contained member in one or more separate processes, and you may nest output from referenced members inside a referring member.

To do this, an alternative format for the VARIABLE clause is provided:

```
VARIABLE Vm TRANSLATE arg USING mp-tr-rule
[PASSING arg [, arg]...]
[clause-2]
```

where *clause-2* is:

```
[IF { arg }  { EQ }  value [AND { arg }  { EQ }  value]...]
    { Vm  }  { NE }             { Vm  }  { NE }
             { GT }                      { GT }
             { LT }                      { LT }
             { GE }                      { GE }
             { LE }                      { LE }
```

where *value* is:

```
{ Dm       }
{ Um       }
{ Sm       }
{ 'string' }
{ NULL     }
```

arg is:

```
{ Dm }
{ Um }
{ Sm }
```

where:

D*m* parameters represent standard attributes and clauses of member definitions.

U*m* parameters represent user-defined attributes (defined by using the User Defined Syntax facility).

S*m* refers to a positional argument passed to this TRANSLATION-RULE from a VARIABLE V*m* translate-clause.

mp-tr-rule is the name of a TRANSLATION-RULE member on the MP-AID.

Only the global parameters and the parameters available to the format line on which the variable appears are available to the clause.

A translate type variable must be specified on a new format line.

The TRANSLATION-RULE for the subordinate members must specify parameter numbers in the form *Sm* to receive the parameters passed. The parameter numbers will be received in the order in which they are passed. For example, processing a format line which contains V1, which is defined as:

```
VARIABLE V1 TRANSLATE D914 USING TRIT1 PASSING D915, D916
```

will cause automatic translation of the item whose name appears in parameter D914, using TRANSLATION-RULE TRIT1.

TRIT1 might, in turn, specify a variable V1:

```
VARIABLE V1 IS D125 IF D110 EQ S1 AND D112 EQ S2
```

`S1` and `S2` will receive the passed parameters D915 and D916 respectively.

`D914` represents the value of an IDMS-SUBSCHEMA buffer size.

`D915` represents the word RESERVE.

`D916` represents the value of the number of areas to be reserved.

`D125` represents the minimum number of digits in an integer.

`D110` represents a form specification.

If you refer to a secondary parameter for which a value is not received, for example, S5 when only three parameters have been passed, a question mark (?) is output in its place.

Refer to "How Variables are Used" on page 21 for the introduction to using variables.

Example: Translation Rule for Hierarchical Structures

An entire data structure can be translated using one TRANSLATE command as shown in the example below.

The GROUP DMRGROUP contains the ITEMS DMRITEM, DMRITEMA, and DMRITEM1.

This translate command is issued:

```
TRANSLATE MANAGER-DICTIONARY MEMBER DMRGROUP
TO OTHER-DICTIONARY
ONTO PUBLIC MPTEMP USING TRGROUP PRINT;
```

In TRANSLATION-RULE TRGROUP (MP-AID name TRGROUP); the following variable statement is specified giving the tr-rule and the parameter (D*n*, S*n*, or U*n*), whose value is the name of the contained or referenced member to be translated:

```
VARIABLE V1 TRANSLATE D317 USING SUB-TRANS PASSING D22, D23 IF D316
EQ 'ITEM'
```

where `D317` is the name of a member contained by the group.

`SUB-TRANS` is the MP-AID name of the translation-rule to be used to translate the contained member. `D22` and `D23` are secondary parameters passed to the nested translation. D316 is the contained member's type.

This nested translate facility then processes the specified member, using the specified tr-rule, to generate input for the specified dictionary.

The nested translate facility also allows the current tr-rule to be used recursively to translate any referenced/contained members of the same type.

```
PRINT OF DMRGROUP
GROUP
ALIAS COBOL 'COBOL-ALIAS-1'
CONTAINS
  (2) DMRITEM
  (DMRITEM) DMRITEMA,
            DMRITEM1
  (19)      DMRITEN2
NOTE 'THE USE OF NAME AND LITERAL BOUND OCCURS'
     'DMRGROUP NOTE STRING 1'

PRINT OF DMRITEM
ITEM
HELD-AS 1 NUMERIC-CHARACTER 3
ALIAS COBOL "ITENCOBALl"
NOTE 'DMRITEM NOTE STRING 1'

PRINT OF DMRITEMA
ITEM
ENTERED-AS 1 PACKED-DECIMAL
SIGNED 10.1
ALIAS COBOL 'DMRITEMA-COBOL-ALIAS-1'
NOTE 'DMRITEMA NOTE STRING 1'

PRINT OF DMRITEM1
ITEM ENTER 1 ALPHA 3
ALIAS COB "COB-DMRITEM"
NOTE 'DMRITEM1 NOTE STRING 1'

PRINT OF TRANSLATION-RULE TRGROUP
TRANSLATION-RULE
FOR MANAGER-DICTIONARY GROUP
MP-AID-NAME TRGROUP
VARIABLE V1 IS 'ELEMENT'    IF D316 EQ 'ITEM'
          ELSE 'STRUCTURE'  IF D316 EQ  'GROUP'
          ELSE 'UNIT'

VARIABLE V2 TRANSLATE D317 USING SUB-TRANS PASSING D22, D23
                         IF 0316 EQ 'ITEM'

VARIABLE V3 TRANSLATE D317 USING TRGROUP PASSING D22, D23
                         IF D316 EQ 'GROUP'
NOTE 'THE NESTED TRANSLATE VARIABLE STATEMENTS WHICH WHEN'
     'RESOLVED WILL TRANSLATE THE CONTAINED MEMBER  (D317),'
     'USING TR-RULE TRITEM or TRGROUP, PASSING THE GROUPS TYPE'
      '(D22) AND NAME (D23) AS SECONDARY PARAMETERS'
VARIABLE V4 IS 'NAME'    IF 0657 NE NULL
            ELSE  'CONSTANT'  IF 0655 NE NULL
```

```
NOTE ’TYPE OF OCCURS BOUND VALUE’
VARIABLE V5 IS   D656 IF D657 NE NULL
            ELSE D654 IF 0655 NE NULL
NOTE ’PICK EITHER THE NAME OR THE LITERAL OCCURS BOUND VALUE’
CONTENTS
FM01 : C15, ’CREATE STRUCTURE ’, D23, ’-’
*
* OTHER-DICTIONARY VERSION OF ADD/REPLACE
*
* OUTPUT ALIASES.
* USE NOSKIP (FN04) TO OUTPUT ’-’ AFTER ALL ALIASES
FM03 :
FM03L : C20, ’ SYNONYM IS ’, D43
*
* OTHER-DICTIONARY VERSION OF ALIAS
*
FN04 : ’-’
* OUTPUT MEMBER-NOTES USING DMR NOTE CLAUSE
*
FM11 : C2O, ’MEMBER-NOTES "’, D49, ’"’
FM12 : C34, ’"’, D49, "’"
FN13 : ’-’
*
* PROCESSING LOOP TO OUTPUT SUB-MEMBER CLAUSES USING
* DMR CONTAINS CLAUSE*
FM192:
FM192L: C2O, ’SUB-MEMBER TYPE ’, V1, ’ IS ’, D317
   - /, C25, ’OCCURENCE IS ’, V4, ’ BOUND ’, V5*
*
* OTHER-DICTIONARY VERSION OF CONTAINS MEMBER
*
* END FIRST LOOP. FMN+2 (FM193)
* WILL CAUSE LOOP BACK TO FMN+1 (FM192)
FM193:
FM01 : C15, ’STRUCTURE END.’
FM192:
*
* TRANSLATE CONTAINED MEMBERS USING NESTED TRANSLATE FEATURE
*
FM192L: /, V2, V3
* THIS IS THE POINT AT WHICH THE CONTAINED MEMBERS ARE
* TRANSLATED USING THE NESTED TRANSLATE FEATURE. THE GENERATED
* LINES FOR THE CONTENTS WILL APPEAR IMMEDIATELY FOLLOWING
* ’STRUCTURE END’
FM193:
FM9999:

PRINT OF TRANSLATION-RULE TRITEM
TRANSLATION-RULE
FOR MANAGER-DICTIONARY ITEM
MP-AID-NAME SUB-TRANS
```

```
VARIABLE V1 IS   'ELEMENT'   IF D22 EQ 'ITEM'
            ELSE 'STRUCTURE' IF D22 EQ 'GROUP'
            ELSE 'UNIT'
VARIABLE V2 IS   'STANDARD-' IF D42 EQ NULL ELSE D42
VARIABLE V3 IS D43 IF D42 EQ NULL
VARIABLE V4 IS   'ELEMENT'   IF S1 EQ 'ITEM'
            ELSE 'STRUCTURE' IF S1 EQ 'GROUP'
            ELSE 'UNIT'
CONTENTS
*
* CREATE ELEMENTS USING DMR MEMBER-TYPE AND VERSIONS
FM01: /, C30, 'CREATE SUB-MEMBER ',V1 , X, D23,'.'
*
* IDENTIFY VERSION/USAGE DETAILS USING FM6O PROCESSING LOOP

*
FM60R1: C35, 'PRIMARY MODE-VERSION IS ',D1l0, X, D112, '-'
*
* NOTE USE OF REPEAT COUNT R1
  - /, C35, 'ELEMENT-USAGE STRING "', D118, '"'
FM61:
*
* IDENTIFY MASTER (CONTAINING), MEMBER USING PASSED PARAMETERS
*
FM01 : C35, 'MASTER-MEMBER TYPE ', V4, ' IS ' , S2, '-'
*
* NOTE USE OF PASSED PARAMETER S2. CALLING GROUP'S NAME
* OUTPUT MEMBER-NOTES USING DMR NOTE CLAUSE
*
FN11 : C35, 'MEMBER-NOTES "', D49, '"'
FM12 : C49, '"', D49, '"'
FN13 : '-'
*
* OUTPUT ALIASES. USE NOSKIP TO OUTPUT '.' AFTER ALL ALIASES
*
FM03:
FM03L: C35, V2, V3, ' SYNONYM IS ', D43
FN04: '-"
*
FM01: C30, 'SUB-MEMBER END.'
FM9999:
```

EXAMPLE OUTPUT FROM TR-RULE TRGROUP

```
CREATE STRUCTURE DMRGROUP.
     COBOL SYNONYM IS COBOL-ALIAS-1
     MEMBER-NOTES "DMRGROUP NOTE STRING 1"
     SUB-MEMBER TYPE ELEMENT IS DMRITEM
     OCCURENCE IS CONSTANT BOUND 2
     .
     SUB-MEMBER TYPE ELEMENT IS DMRITEMA
          OCCURENCE IS NAME BOUND DMRITEM1
     .
     SUB-MEMBER TYPE ELEMENT IS DMRITEM1
     .
     SUB-MEMBER TYPE ELEMENT IS DMRITEM2
          OCCURENCE IS CONSTANT BOUND 19
          .
STRUCTURE END.
```

EXAMPLE OUTPUT FROM TR-RULE TRITEM

```
CREATE SUB-MEMBER ELEMENT DMRITEM.
     PRIMARY MODE-VERSION  IS HELD-AS 1.
     MASTER-MEMBER TYPE STRUCTURE IS DMRGROUP
     MEMBER-NOTES  "DMRITEM NOTE STRING 1"
                   "DMRITEM NOTE STRING 2".
     COBOL SYNONYM IS ITEMCOBAL1.
SUB-MEMBER END.

CREATE SUB-MEMBER ELEMENT DMRITEMA.
     PRIMARY MODE-VERSION IS ENTERED-AS 1.
     MASTER-MEMBER TYPE STRUCTURE IS DMRGROUP.
     MEMBER-NOTES  "DMRITEMA NOTE STRING 1"
     COBOL SYNONYM IS DMRITEMA-COBOL-ALIAS-1.
SUB-MEMBER END.

CREATE SUB-MEMBER ELEMENT DMRITEM1.
     PRIMARY MODE-VERSION IS ENTERED-AS 1.
     MASTER-MEMBER TYPE STRUCTURE IS DMRGROUP.
     MEMBER-NOTES  "DMRITEM1 NOTE STRING 1"
     COBOL SYNONYM IS COB-DMRITEM.
SUB-MEMBER END.
```

An Example TRANSLATION-RULE

This example TRANSLATION-RULE translates an ITEM member type to an ITEM member type. This type of translation, with some adjustments, may be useful when moving members between Manager Products dictionaries (where these dictionaries have some different conventions or requirements).

```
TRANSLATION-RULE
FOR MANAGER-DICTIONARY ITEM
DESCRIPTION
     ’EXAMPLE DYR TRANSLATION-RULE TO TRANSLATE’
     ’MPR PRIMARY DICTIONARY ITEMS.  FOR INPUT TO’
     ’MPR SECONDARY DICTIONARY’
MP-AID-NAME DYRITEM
VARIABLE V1 IS ’ADD’ IF D6 EQ ’STATUS1’
           ELSE ’MODIFY’  IF D6 EQ ’STATUS2’
           ELSE ’REPLACE’
VARIABLE V2 IS ’DICT2-’
VARIABLE V3 IS ’NO MEMBERS’ IF S1 EQ NULL
           ELSE S1
VARIABLE V4 IS ’0’ IF D128 EQ NULL AND 0129 EQ NULL
           AND D124 NE NULL
CONTENTS
* GENERATE ADDIREPLACEIMODIFY
*
FM01: Cl, V1, X, V2, D23, X, ’;’
FM01: C2, D22
*
* GENERATE NOTE CLAUSES. FIRST NOTE STRING =
CONTAINING MEMBER TYPE + IF PASSED
* NAME OF CONTAINING MEMBER
FM11: C6, D48, X, ’"CONTAINED BY ’, V3, X, S1, ’"’
FM12: C2, ’"’, D49, ’"’
*
* GENERATE FORM/VERSION
*
FM60:
FM60L: C3, D110, X D112, X, D124
- /, C3, D117, X, D118, X, D120, X, D122, X, D119, X, D121
- /, C3, D123, X, D125, D126, D127, X, D128, D129, X, V4
- /, C3, D130, X, D131, X, D132
- /, C3, D133, X, D134, X, D135, X, D136, X, D137, X, D138
*
* GENERATE CONTENTS/CONDITION/RANGE-IS
*
FM465:
FM465L: C4, D150, X, D152
- / ,   C4, D155, X, D156
FM468:
FM468L: C5, D153, X, D609, X, D610, X, D611
FM469:
```

```
*
FM471:
FM471L: C5, D159, X, D160, X, D161, D162, X, D163, X, D164
FM472:
*
FM466: FM61:
*
* GENERATE ALIAS INFO
*
FM02: C2, D41, X, D42, X, '"', D43, '"'
FM03: C8, ',', X, D42, X, '"', D43, '"'
*
* GENERATE CATALOG INFO
*
FM05: C2, D44.9, X, '"', D45, '"'
FM06: C4, ',', X, '"', D45, '"'
*
* GENERATE DESCRIPTION
*
FM08: C2, D46, X, '"', D47, V2, '"'
FM09: C13, ',', D47, V2, '"'
*
FM01: ';'
FM9999:
```

3 Export to Another Dictionary

Introduction

DictionaryManager enables you to use a TRANSLATION-RULE to TRANSLATE a dictionary member, and then to TRANSFER the translated member to a file for input to another dictionary.

The translation and transfer capabilities are described according to the identity of the target dictionary. There is an export engine which can be used for a user-selected dictionary, and a selectable unit for the dictionary of particular vendors.

This chapter describes the export engine for User Selected Dictionary (selectable unit DYR-TE00), which is a prerequisite for any export selectable units. With the corporate dictionary definition export engine for User Selected Dictionary, you can export information from a Manager Products dictionary to a dictionary or directory of your choice. This export engine enables you to:

- SET (and QUERY) the default translation rule
- TRANSLATE members to the format of the target dictionary
- TRANSFER the translated members to an external file
- Use a set of format lines and parameter numbers set up for this engine

Refer to "SET and QUERY TRANSLATION" on page 38 for information on the SET and QUERY TRANSLATION command.

How to Translate Members Using the Translation Rules

With the TRANSLATE command, you can translate members from a Manager Products dictionary into source input statements for another dictionary system. You can select an individual member, all members of a member type, or a selection of members of a particular member type.

The translation process will refer to a TRANSLATION-RULE for the member type, supplied by ASG or defined by you, and held on the MP-AID.

The TRANSLATE command and the SET/QUERY TRANSLATION commands use keywords to identify the target dictionary type. The keywords IDD and IDMS are used for translation to the IDD and for IDMS compilers respectively. Use the keyword OTHER-DICTIONARY for translation to a user-selected dictionary.

You can specify which TRANSLATION-RULE is to be used, either as a default in a previous SET TRANSLATION command or to override the default, in the current TRANSLATE command. If you specify an individual member, you must also specify the TRANSLATION-RULE to be used, in a USING clause.

Refer to "SET and QUERY TRANSLATION" on page 38 for the SET and QUERY command.

After translation, the source statements are held in an MP-AID USER-MEMBER. The name you give it must be no more than ten characters long.

The USER-MEMBER can be a PRIVATE USER-MEMBER or a PUBLIC USER-MEMBER. A PUBLIC USER-MEMBER can be accessed by someone with a different logon identifier.

If you have the ControlManager Extended Interactive Facility (selectable unit CMR-FE01), you can edit the USER-MEMBERs.

Refer to "TRANSLATE Command" on page 101 for the syntax of the TRANSLATE command.

Using the TRANSLATE Command

This is the basic command for translating an individual member:

```
TRANSLATE MANAGER-DICTIONARY MEMBER member-name

TO  ⎧ IDD              ⎫ USING mp-tr-rule
    ⎨ IDMS             ⎬
    ⎩ OTHER-DICTIONARY ⎭
ONTO  ⎧ PUBLIC  ⎫ mp-user;
      ⎩ PRIVATE ⎭
```

For example:

```
TRANSLATE MANAGER-DICTIONARY MEMBER EXAMPLE-MEMBER
  TO OTHER-DICTIONARY
  ONTO PUBLIC EX-MEMBER
  USING TR-RULE-X;
```

translates the Manager Products dictionary member EXAMPLE-MEMBER using the MP-AID TRANSLATION-RULE member TR-RULE-X.

This is the basic command for translating the set of all members of a specific member type:

```
TRANSLATE MANAGER-DICTIONARY MEMBER member-name

TO  ⎧ IDD              ⎫
    ⎨ IDMS             ⎬
    ⎩ OTHER-DICTIONARY ⎭
ONTO  ⎧ PUBLIC  ⎫ mp-user;
      ⎩ PRIVATE ⎭
```

For example:

```
TRANSLATE MANAGER-DICTIONARY GROUPS
  TO IDD
  ONTO PUBLIC MP-GROUP ;
```

translates all GROUP member definitions on the Manager Products dictionary to IDD member definition syntax, and outputs the translated definitions to a USER-MEMBER on the MP-AID, named MP-GROUP, which can be copied by someone with a different logon ID. The GROUP members will be translated according to default TRANSLATION-RULEs established by a previously-issued SET TRANSLATION command.

The optional keyword REPLACE is provided so that you can replace the contents of an existing MP-AID member with the output generated by the TRANSLATE command. If REPLACE is not specified, *mp-user* must not be the name of a member that already exists on the MP-AID (unless APPEND is also specified—see below). If REPLACE is specified and *mp-user* does not already exist on the MPAID, a new member named *mp-user* is created.

For example:

```
TRANSLATE MANAGER-DICTIONARY GROUPS
  TO IDMS
  ONTO PUBLIC MP-EXAMPLE REPLACE;
```

The optional keyword APPEND is provided so that you can append the output generated by the TRANSLATE command to the contents of an existing MP-AID member. If APPEND is specified and *mp-user* does not already exist on the MPAID a new member named *mp-user* is created.

For example:

```
TRANSLATE MANAGER-DICTIONARY GROUPS
  TO IDMS
  ONTO PUBLIC MP-EXAMPLE APPEND;
```

If PRINT is specified, all output produced by the TRANSLATE command is printed/displayed as it is generated. If PRINT is omitted, only messages are printed/displayed. PRINT is particularly useful for locating the cause of errors when a TRANSLATE command fails to execute successfully; it enables you to look at the translation process and to see messages in the context of all the output produced.

You can also translate any basic Manager Product member type (such as GROUP), and any user defined members based on that type, by using the keyword GENERIC. To set up user-defined members you need the User Defined Syntax facility.

For example:

```
TRANSLATE MANAGER-DICTIONARY GENERIC GROUP
  TO IDD
  ONTO MP-GROUP PRINT;
```

This will translate all generic GROUP member definitions on the Manager Products dictionary to IDD member definition syntax, and output the translated definitions to a USER-MEMBER on the MP-AID named MPGROUP, automatically printing out the translated source as it does so.

Refer to "SET and QUERY TRANSLATION" on page 38 for the SET and QUERY command and refer to "TRANSLATE Command" on page 101 for the syntax of the TRANSLATE command.

The USING Clause in the TRANSLATE Command

You can apply a TRANSLATION-RULE different from the current default rule by specifying, in an optional clause, the appropriate TRANSLATION-RULE member:

```
USING mp-tr-rule
```

For example:

```
TRANSLATE MANAGER-DICTIONARY GROUPS
  TO IDD
  ONTO PUBLIC MP-GROUP
  USING TR-GROUP;
```

Use of KEEP-DATA Lists in the TRANSLATE Command

You can translate all the members of a member type or generic member type, or you can select members, put them in a KEPT-DATA list, [using KEEP (IN name) (AND)...], and translate those of any one type from there. To do this, insert the keyword KEEP or the clause KEPT IN name:

```
KEEP LIST ONLY PROJ07;
TRANSLATE MANAGER-DICTIONARY KEPT GROUPS TO IDD
ONTO PRIVATE P07-MP-GRP USING TR-GRP-P07
```

This translates any GROUP member definitions in the Manager Products unnamed KEPT-DATA list to IDD member definition syntax, according to the TRANSLATION-RULE TR-GRP-P07. It then stores the translated definitions to a USER-MEMBER named P07-MP-GRP on the MP-AID. The USER-MEMBER can only be used through your logon ID. The TRANSLATION-RULE TR-GRP-P07 will override any default established by a previously issued SET TRANSLATION command.

How to TRANSFER Members

The TRANSFER command enables you to transfer translated dictionary member definitions from the MP-AID to an external file. Since the definitions will be in the form of source statements for another type of dictionary, you can use the external file as input to that dictionary. This is the basic form of the command:

```
 TRANSFER
    FROM USER-MEMBER mp-user [LOGON logon-id]
    TO FILE filename ⎧ PARTITIONED                  ⎫
                     ⎩ SEQUENTIAL ['control-card']⎭
    [AS library-name]
```

For example:

```
TRANSFER FROM USER-MEMBER MP-GROUP
TO FILE DIC2SRC PARTITIONED;
```

This transfers the member definitions, previously translated to source for the target dictionary, from the MP-AID USER-MEMBER MP-GROUP to an output source library dataset with a logical file name of DIC2SRC. (This is the ddname or dtfname used in job control statements.)

The TRANSFER command can optionally include the keywords PARTITIONED and SEQUENTIAL for particular environments. An AS clause is also available to generate your own library name.

Refer to "TRANSLATE Command" on page 101 for the syntax of the TRANSFER command.

Keywords PARTITIONED and SEQUENTIAL in the TRANSFER Command

The keyword PARTITIONED is available for OS environments. It defines the dataset to be a BPAM-partitioned dataset.

The alternative keyword SEQUENTIAL defines the dataset to be a QSAM sequential dataset. With this alternative, you have the option of adding a library system control card image to the output dataset, immediately before the copied source statements.

This is a character string of up to 72 characters in quotes, which may contain a single question mark (?) to indicate the point at which the generated library member name is to be inserted in the control card. If your character string occupies less than 72 characters, trailing spaces are implied.

For example:

```
TRANSFER FROM MEMBER-NAME MP-GROUP
      TO FILE IDDSRC SEQUENTIAL ' CATALS X.?'
```

You can specify a name to be generated as the library name, in an optional AS clause. Otherwise the MP-AID name is used.

If the output dataset is SEQUENTIAL and you do not specify a control card, then the standard IBM library update control card is used, as follows:

- Under OS, an IEBUPDTE control card:

  ```
  './ ADD LIST=ALL,NAME=?'
  ```

- Under DOS, a MAINT control card:

  ```
  CATALS I.?'
  ```

where the question mark (?) indicates the point at which the generated library member name is inserted. The control card is written to the output dataset, immediately before the copied source statements.

With the above form of the command, the copied source statements are catalogued in the output source library dataset with the name of the MP-AID USER-MEMBER being transferred.

Use of the AS Clause in the TRANSFER Command

You can command generation of your own library name, by adding the optional clause:

```
AS library-name
```

For example, the command:

```
TRANSFER FROM USER-MEMBER MP-GROUP
             TO FILE DIC2SRC AS #MPRGRUP
```

will transfer the member definitions translated to source for the target dictionary from the MP-AID USER-MEMBER MP-GROUP to an output source library dataset with a logical file name (that is the ddname or dtfname used in job control statements) of DIC2SRC.

The generated library member in the output data has the name ƒMPRGRUP. The name must not be more than 16 characters long. The first character must be alphabetic, ƒ, £, %, @, or a local currency symbol with the internal code hexadecimal SB.

SET and QUERY TRANSLATION

With DictionaryManager installed, Manager Product dictionary members can be defined, translated, and transferred to dictionaries of other products. The rules that describe the detailed equivalencies between the Manager Product dictionary and the target dictionary are set out in a DictionaryManager translation rules.

The SET TRANSLATION command is used to:

- SET the default rule that will be used when a member of a particular type is to be translated to the specified dictionary.
- Specify the OPTIONS that will be output (once only) at the beginning of every translate session (for IDD).

Refer to the *ASG-ControlManager User's Guide* for details of how to tailor your environments.

SET and QUERY TRANSLATION for Base Member Types

To set a default TRANSLATION-RULE for a particular member type, enter:

```
SET TRANSLATION [ { OTHER-DICTIONARY }
                  { IDD              }
                  { IDMS             }
TR-RULE FOR MANAGER-DICTIONARY member-type
TO mp-tr-rule;
```

where:

member-type is a base member type name such as GROUP or ITEM.

mp-tr-rule is the name of the TR-RULE member on the MP-AID that is to be used in the TRANSLATION.

The TR-RULE clause can be repeated.

OTHER-DICTIONARY is the keyword for use with the corporate dictionary definition export for User Selected Dictionary.

IDD is the keyword to use for translation to the IDD compiler; IDMS is the keyword to use for translation to the IDMS compiler. Export to IDD and IDMS is possible only if the corporate dictionary definition for export to DD selectable unit DYR-TE08 is installed.

To find out the current TRANSLATION setting, enter:

```
QUERY TRANSLATION [ { OTHER-DICTIONARY }
                    { IDD              }
                    { IDMS             }
```

SET and QUERY TRANSLATION for UDS Member Types

To set a TRANSLATION-RULE for a Manager Product UDS member type, the User Defined Syntax facility must be installed.

To set a TRANSLATION-RULE for a Manager Product UDS member type, enter:

```
SET TRANSLATION [ { OTHER-DICTIONARY }
                  { IDD              }
                  { IDMS             }
TR-RULE FOR MANAGER-DICTIONARY member-type
TO mp-tr-rule;
```

where:

member-type is a UDS member type such as PROCESS.

mp-tr-rule is the name of the TR-RULE member on the MP-AID that is to be used in the TRANSLATION.

The TR-RULE clause can be repeated.

OTHER-DICTIONARY is the keyword for use with the corporate dictionary definition export for User Selected Dictionary.

IDD is the keyword to use for translation to the IDD compiler; IDMS is the keyword to use for translation to the IDMS compiler. Export to IDD and IDMS is possible only if the corporate dictionary definition for export to IDD is installed.

If the User Defined Syntax facility is installed, then as well as setting the translation of the basic ASG-supplied member type, you can also set the translation of all user-defined member types based on that type; this is done by including the keyword GENERIC in the TR-RULE clause.

```
   SET TRANSLATION [ { OTHER-DICTIONARY }
                     { IDD              }
                     { IDMS             }
   TR-RULE FOR MANAGER-DICTIONARY GENERIC member-type
   TO mp-tr-rule;
```

For example:

```
SET TRANSLATION OTHER-DICTIONARY
    TR-RULE FOR MANAGER-DICTIONARY GENERIC GROUPS
    TO TR-GGRP
```

SET and QUERY TRANSLATION for Cullinet’s IDD

With DictionaryManager installed, Manager Product dictionary members can be defined, translated, and transferred to Cullinet’s IDD. The rules that describe the detailed equivalencies between the Manager Product dictionary and IDD are set out in translation rules. A set of default rules is supplied by ASG.

For example, to set up default TRANSLATION-RULEs:

- For a Manager Product member type ITEM and for all user-defined member types based on ITEM
- For a user-defined member type PROCESS

enter:

```
SET TRANSLATION IDD
TR-RULE FOR MANAGER-DICTIONARY GENERIC ITEM TO DEF-TR-ITM
TR-RULE FOR MANAGER-DICTIONARY PROCESS TO DEF-TR-GGP;
```

You can also specify sign on and session options for the IDD dictionary. For example:

```
SET TRANSLATION IDD
SIGNON ’USER FRED DICTIONARY IDD1’,
’USAGE MODE IS PROTECTED UPDATE’
OPTIONS ’QUOTE IS "’,
’INPUT COLUMNS ARE 1 THRU 72’;
```

The QUERY TRANSLATION command can be used as follows:

```
QUERY TRANSLATION;
```

causes all the previously SET TRANSLATION-RULES to be displayed.

```
QUERY TRANSLATION IDD;
```

causes the previous SET IDD TR-RULES, SIGNON and OPTIONS to be displayed.

```
QUERY TRANSLATION IDD SIGNON;
```

causes the previously SET IDD SIGNON information to be displayed.

```
QUERY TRANSLATION IDD TRANSLATION-RULES;
```

causes the previously SET IDD TR-RULES to be displayed.

4 Import Executive Routines and Variables

Extract Executive Routines

Introduction

Extract executive routines are used during the extract stage of the import procedure. They are invoked with the USING keyword of the EXTRACT command.

When importing objects from an external environment (for example, a COBOL program) into a repository, you should use extract executive routines to reorganize extracted information. This information can be used in the reconcile and preview stages of the import procedure.

You can extract information for other purposes: building reports, for example, or taking advantage of the Procedures Language to extract, reformat, and then export information (without adding it to a repository). For such purposes, the subsequent import stages are not required and the purpose of extract executive routines is simply to manipulate extracted information.

The following sections assume that you are extracting external objects in order to import them into a repository; however, if you intend to extract for other purposes you may still find them useful.

Whatever your purpose in extracting information from an external environment, it will help if that environment is subject to standards. This makes it simpler to write extract executive routines to locate required information.

Storing Extracted Information

Extracted information can be copied into:

- A single default variable array called EXT_RECORD. Each record read is extracted whole, and stored in a separate element of the array.
- One or more named variable arrays. You can store specific fields of records in different variables and select or exclude data meeting certain conditions using the data storage and selection facilities of the EXTRACT command.

For example, the following might be stored in the first five elements of EXT_RECORD when extracting from a COBOL program:

Element Number	Extracted Contents
1	`'*-----------------------'`
2	`'IDENTIFICATION DIVISION.'`
3	`'*-----------------------'`
4	`'SKIP2'`
5	`'PROGRAM-ID.                SDD11.'`

Note:

The variable EXT_RECORD is automatically cleared before being used. If you specify your own variables to hold data, you must clear them yourself; if they are not cleared, data in them is overwritten.

After storing extracted data, you can then use extract executive routines to move information into other variables. These can be:

- Mandatory variables to be used by executive routines during the reconcile stage
- User-defined variables (that is, variables of your choice), to be used by executive routines to derive the repository definitions you want, during the preview stage

You must populate the mandatory variables, but use of user-defined variables is optional. You may be able to build preview executive routines that manipulate information in the way you want without moving it out of the default array EXT_RECORD.

For more complex translation, it is usually more productive to store information in other variables, either using extract executive routines or the data selection and storage capabilities of the EXTRACT command.

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for details of the EXTRACT command.

For complex translations, the data storage and selection facilities of the EXTRACT command may not be sufficient. In these cases, you should write extract executive routines to move extracted information stored on the WBTA as required.

You can use a combination of the SEARCH, WORD, and SUBSTR directives to search for a particular string or keyword (and associated subordinate strings/keywords) and then move selected information into other variables.

For example, if you are extracting the source of a COBOL program, you might want to extract information from CALL and associated USING statements.

If you know the row and character position of a string within an array, you can use the SUBSTR directive to copy it to another variable. For example, to put the contents of the 10th to 18th character positions of the third element in EXT_RECORD into the third element of the variable PROG_NAME, enter:

```
PROG_NAME(3) = SUBSTR(EXT_RECORD(3),10,18)
```

When you have extracted free-form (as opposed to fixed-form) information, the WORD directive will be useful. For example:

```
if WORD (EXT_RECORD(a), 3) = :USING: then do
```

(For the purpose of the examples used in this section, assume that the colon character (:) has been defined as a literal delimiter).

Both WORD and SUBSTR can be combined with SEARCH to scan for a string and store parts of it in a variable. For example:

```
ROWI = SEARCH(:EXT_RECORD:,:SEARCH-STRING:,,,:P:)
IF ROWI > 0 THEN VARIABLE=WORD(EXT_RECORD(ROWI),2)
```

How to Organize Information on the WBTA

You should organize extracted information into variable arrays (for both mandatory and user-defined variables) as follows:

- Use dedicated arrays; that is, store occurrences of the same type of information in the same array.
- Use dedicated elements; that is, use the same element in each of the dedicated arrays to store information that relates to one object.

The following table represents the method described above, using two mandatory variables (EXT_OBJ_NAME and EXT_OBJ_TYPE) and two user-defined variables (EXT_SECURITY and EXT_DESCRIPTION) as examples of how information relating to one object (the third on the WBTA) should be stored.

Array (element)	**Contents**
EXT_OBJ_NAME (3)	'PG-SDD11'
EXT_OBJ_TYPE (3)	'PROGRAM'
EXT_SECURITY (3)	'200'
EXT_DESCRIPTION (3)	'EXAMPLE'

All of the extracted information relating to a program called SDD11 is stored in the third element of each array. Each array is used to hold all occurrences of a particular type of information.

Mandatory Variables

Introduction

Before building extract executive routines you must consider what part of the extracted information you want to translate into a repository member and what relationships each of these repository members are to have with one another.

The information to be stored in the mandatory variables is related to the identity of the repository members you want to derive from the extracted information, the relationships between them, and whether or not you want a definition to be generated. The information must be stored in mandatory variables, to be located by reconcile executive routines.

Parents, Children, and Referenced Objects

An external object imported during the extract stage is known as a parent object, and can refer to other objects (known as its children). You can use the information imported about a parent object to generate a repository member that contains attributes defining the parent object's relationship to its children.

Children that can have children of their own are known as referenced objects.

Information about children that are not referenced objects can be extracted and then documented as repository members. You cannot extract information about referenced objects.

During the populate stage, referenced objects are added to the repository as dummy members referenced by the member documenting the parent object, unless members with these names already exist in the repository (in which case, relationships are created from the parent to these members).

To show parent, children, and referenced objects, use the reconciliation report produced by the RECONCILE command.

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for details of the RECONCILE command.

Parent-children Relationships

The reconciliation report groups objects together in a parent-child chain. You should decide if any of the relationships you want to establish can be usefully represented in this report, in terms of a parent-child relationship.

If you are importing a number of tables and associated objects from DB2, the reconciliation report displays each imported table as parents followed by the objects associated with it (for example, the columns) as children. This makes the reconciliation report easier to read, since each table and its children are grouped together.

However, if you import a message library from ISPF, the parent-child relationship may not be relevant; you intend each message to be a separate repository member without references to the others. In such a case all the imported objects might best be regarded as parents.

Create parent-children relationships using the variables EXT_OBJ_CHAIN and EXT_OBJ_PARENT_POINTER.

Parents and children need to be chained together via EXT_OBJ_CHAIN; the parent points to one child, which in turn points to another, and so on. Each child must point to the parent via EXT_OBJ_PARENT_POINTER. Each extracted object may have several parents and each parent may have any number of children.

The relationships represented in the reconciliation report are not automatically translated into relationships in the repository. However, you can use the information stored in EXT_OBJ_CHAIN to build a reference (a CONTAINS attribute, for example) into the definition generated by your preview executive routines for a parent object. You can also manipulate the information stored in EXT_OBJ_PARENT_POINTER to build references (via SEE attributes for example) between the definitions generated for the children.

You may not want to generate definitions for all extracted objects. You may want to generate definitions for some objects, but only references to dummy members for others.

You can control the generation of definitions using the DMR_MEM_GEN variable. Set this to GEN if you want your preview executive routines to generate a definition, or to REF if you want your preview executive routines to generate a reference only.

Example of a Parent-child Relationship

Following is an example of how a parent-child relationship can be created for the objects named (in EXT_OBJ_NAME) Parent, Child1, Child2, and Child3, with the variables EXT_OBJ_CHAIN and EXT_OBJ_PARENT_POINTER.

This table gives values for the variables named above:

Object	EXT_OBJ_NAME	EXT_OBJ_CHAIN	EXT_OBJ_PARENT_POINTER
a	Parent	b	null
b	Child1	c	a
c	Child2	d	a
d	Child3	null	a

Figure 3 on page 48 illustrates the structure described above:

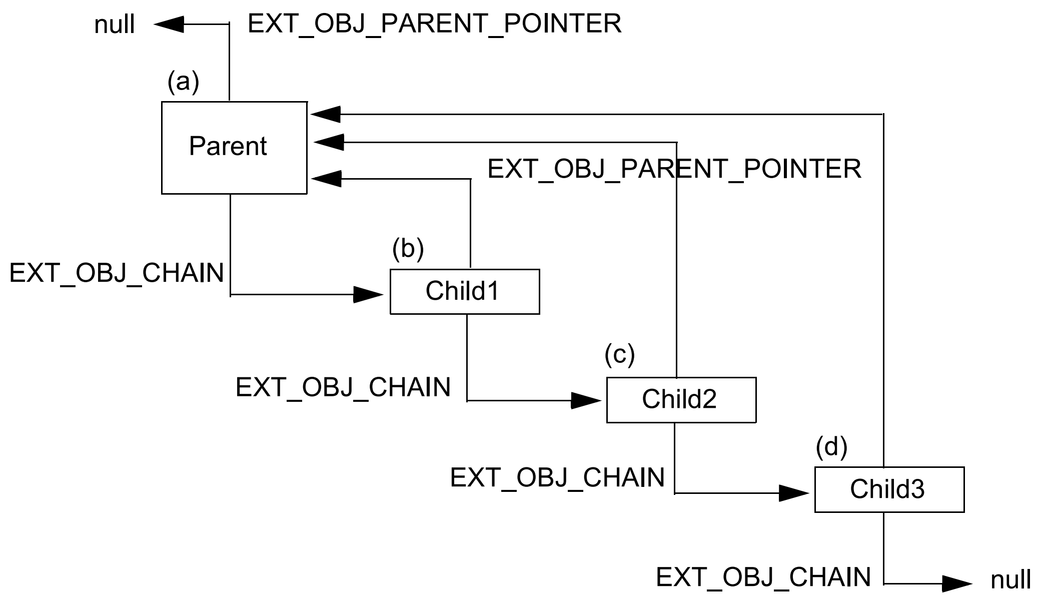


Figure 3. Example of a Parent-child Relationship

Note:

Null equals no value. To set an element of a variable to no value, enter:

```
variable_name(x)=::
```

where `variable_name` is the name of the variable, and `x` is the element to be set to null.

User-defined Variables

Introduction

During the extract stage, you should structure the extracted information so that your preview executive routines can find and process it efficiently. You can do this using extract executive routines and/or the data selection and storage capabilities of the EXTRACT command.

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for details of the EXTRACT command.

Repeating Attributes

To store information about repeating attributes (that is, multiply occurring attributes—a DESCRIPTION attribute with several lines, for example) you must define these items:

- A POINTER variable you can relate to the object
- An OCCURRENCE variable you can relate to the object
- A variable array to contain the repeating attribute

For example, to store four description lines of program SDD11:

- Define the variables EXT_OBJ_DESC_PTR (the pointer), EXT_OBJ_DESC_OCC (the occurrence), and PGM_DESC (the new array to hold description lines).
- Set EXT_OBJ_DESC_PTR(c) to one, to show that the pointer belongs to SDD11 (object c). The value it is set to is the first element number within PGM_DESC which holds a description line (element 1).
- Set EXT_OBJ_DESC_OCC(c) to four, to show a total of four description lines
- Set PGM_DESC(1) to description line one of SDD11, PGM_DESC(2) to description line 2 of SDD11, and so on.

This arrangement is shown in Figure 4 on page 49.

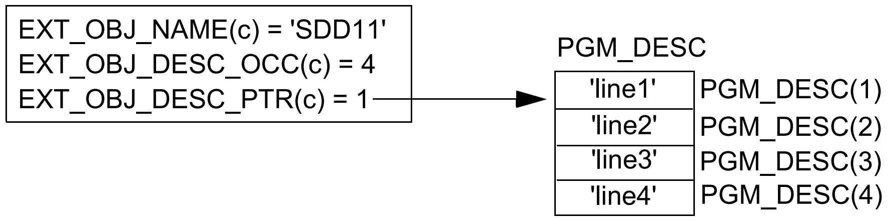


Figure 4. Example of Storing Repeating Attributes

Relationships

You can use a related group of arrays to store information about relationships between objects. For example, to store information about which modules program SDD11 calls and the relevant CALL parameters, define these variables:

- EXT_OBJ_REL_PTR (a pointer)
- EXT_OBJ_REL_OCC (the occurrence)
- CALLS_PARM (to hold the CALLS parameters and used by preview executive routines to form a CALLS attribute in the repository)
- CALLS_MOD_PTR (a pointer to the modules that SDD11 calls)
- CALLS_MOD_OCC (the number of calls that SDD11 includes)

Figure 5 on page 50 represents the method described above:

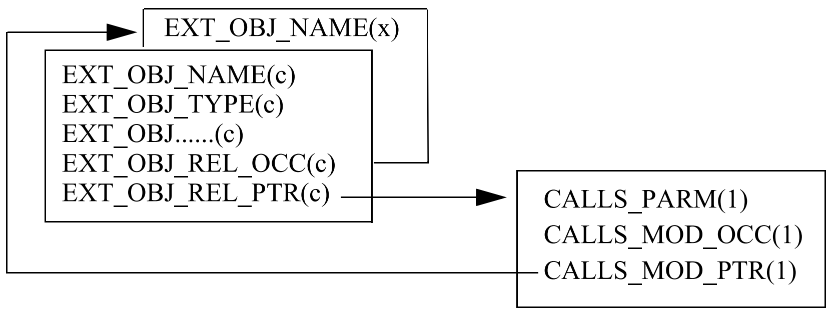


Figure 5. Example of Storing Relationship Information

So, the related group of variables CALLS_PARM, CALLS_MOD_OCC, and CALLS_MOD_PTR constitute the relationship between the program SDD11 (object c) and the modules that it calls (object x, for example).

Reconcile Executive Routines

Introduction

Reconcile executive routines are used during the reconcile stage of the import procedure, and are invoked via the USING keyword in a RECONCILE command.

Reconcile executive routines are used to generate proposed member names and types for your consideration before you generate definitions and subsequently add them to a repository.

Use reconcile executive routines to process extracted information as follows:

- Translate the information stored in EXT_OBJ_TYPE to represent specified repository member types and store this translated information in DMR_MEM_TYPE.
- Translate the information stored in EXT_OBJ_NAME to represent specified repository member names and store this translated information in DMR_MEM_NAME.

Element numbers of variables holding information to be translated should match up with element numbers of variables storing translated information.

For example, after examining and translating the contents of EXT_OBJ_NAME(2), the translated information should be stored in DMR_MEM_NAME(2).

A reconciliation report is automatically displayed during the reconcile stage. This report gives details of the proposed members' names and types and whether or not they are already present in the repository.

Refer to Chapter 6, "Syntax of Commands and Member Types," on page 67 for details of the RECONCILE command.

Tailoring Common Clauses

After reconcile executive routines are executed, the executive routine MPDYWTEXCC is invoked. If any proposed member names already have a definition in the currently open repository (that is visible from the current status), this routine extracts the common clauses from the repository members and copies them into variables on the WBTA. These clauses can subsequently be merged by preview executive routines with the proposed members.

You can alter MPDYWTEXCC to tailor the way that common clauses are handled. For example, if your installation standards mean that you do not need COMMENT attributes, you can tailor MPDYWTEXCC to suppress it.

MPDYWTEXCC is passed two parameters:

- *`curr`* is the DMR_MEM_NAME array number of the current member being processed.
- *`head`* is the DMR_MEM_NAME array number of the first member in a chain of members with the same name.

`head` equals *`curr`* when the current member is processed for the first (or only) time. If the two values differ, then the current member (at *`curr`*) has already been processed (at *`head`*) and the variables associated with *`head`* can be used for *`curr`*; so, no more DRETRIEVEs are needed.

To simplify common-clause processing, rules which drive the processing are set up in the supplied executive routine MPDYWTDFLT, called at the start of the reconcile stage. These rules can be altered or added to as shown in the routine. Alternatively, you can add your own, special processing, as shown in MPDYWTEXCC.

Mandatory Variables

Certain variables must be used to store particular types of extracted information and other information required during the reconcile stage of the import procedure (by the RECONCILE command).

The mandatory variable names, and the type of information that should be stored in each, are listed in this table.

Variable Name	Stores
EXT_OBJ_NAME	The name of an imported object in the environment from which it was imported.
EXT_OBJ_ID	The fully qualified name of an imported object in the environment from which it was imported. This variable supports the Manager Products message DM05703I that is output as the result of an EXTRACT command to tell you the names of the external objects that have been imported.
EXT_OBJ_TYPE	The type of the imported object in the environment from which it was imported.
EXT_OBJ_PARENT_POINTER	A pointer from a child to a parent object (in the context of the parent child relationship represented in the reconciliation report produced by the RECONCILE command).

Variable Name	Stores
EXT_OBJ_CHAIN	A pointer to the next child in a parent-child relationship chain.
EXT_OBJ_CHAIN_END	Pointer from the parent object to the last child in a parent-child relationship chain.
DMR_MEM_GEN	Indicates if a member definition is to be generated (GEN) or used as a reference only (REF).
EXT_OBJ_OCC	The number of imported objects (not including references).

Preview Executive Routines

Introduction

Preview executive routines are used during the preview stage of the import procedure, and are invoked via the USING keyword in a PREVIEW IMPORT command.

Preview executive routines translate the information stored on the WBTA (during the extract and reconcile stages) into repository definitions in the format you require.

You can write a line of definition by using the WRITEF directive to copy information stored in variables into a definition. This definition may subsequently be used to update a repository during the populate stage (via the POPULATE command).

System-generated Variables

These variables are generated automatically during the import process:

Variable Name	Use
EXT_SAVE_GEN	Contains a copy of DMR_MEM_EN as generated by EXTRACT. Used for performance reasons.
EXT_STAGE	Describes the import stage reached: extract, reconcile, or preview.
MPDY_EXTRACT_SOURCE	Set by the EXTRACT command to describe the environment from which the extract was made: EXTERNAL-FILE, COMMAND, VARIABLE, DB2, or SQL/DS.
REC_REP_TYPE	Set by the RECONCILE command to describe what type of report has been requested: SUMMARY or DETAIL.

5 Format Lines and Parameter Numbers

Format Lines and Parameter Numbers Common to All Member Types

These format lines and parameter numbers are common to all ASG-supplied member types.

Format line number groups may be indented to indicate that the related clause is subordinate. The indented group is subordinate to the preceding group that is not indented as far. Three levels of indentation are used. A number from an indented group can only be used within a processing loop for its superior clause(s).

Details of System Parameters

Format Line Numbers	Parameter Number	Description
FM01	D1	Date
	D2	Dictionary user-name
	D3	Time
	D4	Logon ID
	D5	Dictionary name
	D6	Status name
	D7	Status condition

These are other parameter numbers that may be used in any format line:

Format Line Numbers	Parameter Number	Description
	D21	'REPORT OF'
	D22	Member type
	D23	Member name
	D24	'EDITION'
	D25	Edition number / **DUMMY**

Format Line Numbers	Parameter Number	Description
	D26	'ENCODED BY'/'CREATED-BY'
	D27	User
	D28	'AT'
	D29	Time
	D30	'ON'
	D31	Date
	D32	'STATUS'
	D33	Status name
	D34	'PROTECTION'
	D35	'NONE'/'OWNER'
	D36	'REMOVE'
	D37	'ALTER'
	D38	'ACCESS'
	D39	'OWNED BY'
	D40	Owner name
FM02, FM03, FM04	D41	'ALIAS'
	D42	Alias type
	D43	Alias name
FM06, FM06, FM07	D44	'CATALOGUED AS'
	D45	Catalog classification string
FM08, FM09, FM10	D46	'DESCRIPTION'
	D47	Description string
FM11, FM12, FM13	D48	'NOTE'
	D49	Note string
FM14, FM15, FM16	D51	'ADMINISTRATIVE-DATA'
	D52	Administrative data string
FM32, FM33, FM34	D77	'EFFECTIVE-DATE'
	D78	Effective date
FM35, FM36, FM37	D79	'OBSOLETE DATE'
	D80	Obsolete date
FM38, FM39, FM40	D53	'COMMENT'

Format Line Numbers	Parameter Number	Description
	D54	Comment string
FM41, FM42, FM43	D55	'QUERY'
	D56	Query string
FM44, FM45, FM46	D57	'FREQUENCY'
	D58	Process mode
	D59	Interval
	D60	Value/details
FM47, FM48, FM49	D70	'SECURITY-CLASSIFICATION'
	D71	Security details string
	D72	'FROM'
	D73	Date
FM50, FM51, FM52	D74	'ACCESS-AUTHORITY'
	D75	Access authority
	D76	Name of access authority mode
FM53, FM54, FM55	D81	'SEE'
	D82	See member name
	D83	'FOR'
	D84	See qualification
A concluding line for the whole output:		
FM9999		Any character string

Format Lines and Parameter Numbers for Basic Member Types

For ITEM Members

Format line number groups may be indented to indicate that the related clause is subordinate. The indented group is subordinate to the preceding group that is not indented as far. Three levels of indentation are used. A number from an indented group can only be used within a processing loop for its superior clause(s).

Format Line Numbers	Parameter Number	Description
FM59, FM60, FM61	D110	Form
	D111	'VERSION'
	D112	Version number
	D113	'USER-EXIT'
	D114	User exit module name
FM461, FM462, FM463	D115	'HEADINGS'
	D116	Heading string
FM59, FM60, FM61	D117	'USAGE'
	D118	Usage string
	D119	'SIGNED'/'UNSIGNED'
	D120	'VARIABLE'/'FIXED'/'NULL'
	D121	'TRUNCATED'/'ROUNDED'
	D122	'COMPRESSED'
	D123	'LEFT-'/'RIGHT-JUSTIFIED'
	D124	Item type
	D125	Minimum length integer digits
	D126	Minimim length fraction digits
	D127	'TO'
	D128	Integer digits
	D129	Fraction digits
	D130	'NAME'
	D131	Item name
	D132	Item version
	D133	'PICTURE'
	D134	Picture string (delimited)

Format Line Numbers	Parameter Number	Description
	D135	'WITH'
	D136	'SEPARATE'
	D137	'LEADING'/'TRAILING'
	D138	'SIGN'
FM464, FM465, FM466	D150	'CONTENTS'
	D151	'ELSE'
	D152	Item type ('ALPHABETIC'/'ALPHAMERIC'/.../'FLOATING -POINT')
	D155	'CONDITION-NAME'
	D156	Condition name
	D157	'FORMA'
	D158	Format
FM467, FM468, FM469	D153	'IS'/'RANGE'
	D609	Lower limit
	D610	'TO'
	D611	Upper limit
FM470, FM471, FM472	D159	'IF'/'AND'/'OR'
	D160	Item name B
	D161	Version B
	D162	Operator ('EQ'/.../'LE')
	D163	Item name C/literal
	D164	Version C

For GROUP Members

Format Line Numbers	Parameter Number	Description
FM185, FM186, FM187	D300	'FORM'
	D301	Form (ENTERED/HELD/REPORTED/ DEFAULTED-AS)
	D302	'USER-EXIT'
	D303	User exit name
FM188, FM189, FM190	D304	'KEYS'
	D305	Member name
	D306	'UNIQUE'/'DUPLICATED'
	D307	'ASCENDING'/'DESCENDING'
FM191, FM192, FM193	D310	'CONTAINS'
	D311	'NO MEMBERS'
	D312	'ELSE'
	D313	Bound start - '('
	D314	Bound literal - name or numeric
	D315	Bound version
	D654	Bound literal - numeric
	D655	')' following numeric
	D656	Bound literal - name
	D657	')' following name
	D316	Member type
	D317	Member name
	D318	Member version
	D319	'ALIGNED'/'UNALIGNED'/'NOT-ALIGNED'
	D320	'KNOWN-AS'
	D321	Known-as name
FM476, FM477, FM478	D330	'IF'/'AND'/'OR'
	D331	Item name B
	D332	Version B
	D333	Operator ('EQ'/.../'LE')
	D334	Item name C/literal
	D335	Version C

For FILE Members

Format Line Numbers	Parameter Number	Description
FM62, FM63, FM64	D180	'GENERATION-CYCLE'
	D181	Generation cycle
FM65, FM66, FM67	D182	'RETENTION PERIOD'
	D183	Retention period value
	D184	Retention period detail units
FM68, FM69, FM70	D185	'GROWTH-RATE'
	D186	Growth rate value
	D187	'PERCENT'
	D188	Details string/interval
FM71, FM72, FM73	D173	'DENSITY'
	D174	Density
FM74, FM75, FM76	D175	'VOLUME'
	D176	Volume
FM77, FM78, FM79	D177	'SIZE'
	D178	Size value
	D179	Size—units/size—details (STRING)
FM80, FM81, FM82	D190	'ORGANIZATION'
	D191	'IS'
	D192	'SEQUENTIAL'
	D193	'INDEXED'
	D194	'DIRECT'
	D195	'KEYED'
	D196	'VSAM'
	D197	'PARTITIONED'
FM83, FM84, FM85	D202	'BLOCKING'
	D203	'IS'
	D204	Block format
FM86, FM87, FM88	D205	'BLOCK SIZE'
	D206	'IS'

Format Line Numbers	Parameter Number	Description
	D207	Block size
FM89, FM90, FM91	D208	'RECORD SIZE'
	D209	'IS'
	D210	Record size
FM92, FM93, FM94	D198	'NO LABELS'/'LABELS'
	D199	'ARE'
	D200	'STANDARD LABELS'/user-labels module
	D201	User-label module name
FM98, FM99, FM100	D170	'DEVICE'
	D171	Device type
	D172	Device number/model
FM101, FM102, FM103	D211	'SORT KEY'
	D212	Member type
	D213	Member name
	D214	'ASCENDING'/'DESCENDING'
FM185, FM186, FM187	D300	'FORM'
	D301	Form (DEFAULTED AS/ENTERED/HELD/ REPORTED)
FM191, FM192, FM193	D310	'CONTAINS'
	D311	'NO MEMBERS'
	D312	'ELSE'
	D313	Bound start '('
	D314	Bound literal name or numeric
	D315	Bound version
	D654	Bound literal numeric
	D655	')' following numeric
	D656	Bound literal name
	D657	')' following name
	D316	Member type
	D317	Member name
	D318	Member version

Format Line Numbers	Parameter Number	Description
	D319	'ALIGNED'/'UNALIGNED'
	D320	'KNOWN-AS'
	D321	Known-as name
FM476, FM477, FM488	D330	'IF'/'AND'/'OR'
	D331	Item name B
	D332	Version B
	D333	Operator ('EQ'/.../'LE'
	D334	Item name C/literal
	D335	Version C

For MODULE, PROGRAM, and SYSTEM Members

Details of PROGRAM Members

Format Line Numbers	Parameter Number	Description
FM110, FM111, FM112	D215	'AUTHOR'
	D216	Author
FM113, FM114, FM115	D217	'INSTALLATION'
	D218	Installation name
FM116, FM117, FM118	D219	'DATE-WRITTEN'
	D220	Date
FM119, FM120, FM121	D221	'SOURCE-COMPUTER'
	D227	Source computer
FM122, FM123, FM124	D228	'OBJECT-COMPUTER'
	D229	Object computer
FM125, FM126, FM127	D230	'SPECIAL NAMES'
	D231	Special names
FM128, FM129, FM130	D232	'I-O-CONTROL'
	D233	IO control (TXT)
FM131, FM132, FM133	D234	'ASSIGNMENT'
	D235	Assignment string
FM134, FM135, FM136	D236	'EDIT-INPUT'
	D237	Edit input (TXT)
FM137, FM138, FM139	D238	'EDIT-OUTPUT'
	D239	Edit output string
FM140, FM141, FM142	D240	'EDIT-UPDATE'
	D241	Edit update string

Details of MODULE, PROGRAM, and SYSTEM Members

Format Line Numbers	Parameter Number	Description
FM146, FM147, FM148	D250	'LANGUAGE'
	D251	Language string/'NOT SPECIFIED'
FM149, FM150, FM151	D253	'ENTRY POINT'
	D254	Entry point string
FM152, FM153, FM154	D255	'PARAMETERS'
	D256	'NO-DATA'
	D257	Member type
	D258	Member name (for parameters)
FM155, FM156, FM157	D260	'INPUTS'
	D262	Member type
FM158, FM159, FM160	D264	'OUTPUTS'
	D266	Member type
	D267	Member name
FM161, FM162, FM163	D268	'UPDATES'
	D270	Member type
	D271	Member name
FM164, FM165, FM166	D275	'CALLS'
	D276	'NO MEMBERS'
	D277	Member type
	D278	Member name
	D279	'AT'
	D280	At string (entry point)
FM167, FM168, FM169	D281	'PASSING'
	D282	Member type passed
	D283	Member name passed
FM173, FM174, FM175	D284	'PROCESSES'
	D285	DB type
FM176, FM177, FM178	D290	'CONTAINS'

Details of MODULE, PROGRAM, and SYSTEM Members

Format Line Numbers	Parameter Number	Description
	D291	'NO MEMBERS'
	D292	Member type
	D293	Member name

Format Lines and Parameter Numbers for ASG-DesignManager Members

Format line number groups may be indented to indicate that the related clause is subordinate. The indented group is subordinate to the preceding group that is not indented as far. Three levels of indentation are used. A number from an indented group can only be used within a processing loop for its superior clause(s).

Details of ENTITY Members

Format Line Numbers	Parameter Number	Description
FM479, FM480, FM481	D613	'IDENTIFIER'
	D614	'IS'
	D615	Member name
FM482, FM483, FM484	D616	'ONE-ATTRIBUTES'
	D617	'ARE'
	D618	Member name
FM485, FM486, FM487	D619	'MULTI-ATTRIBUTES'
	D620	'ARE'
	D621	Member name
FM488, FM489, FM490	D622	'ONE-ASSOCIATIONS'
	D623	'TO'
	D624	Member name
FM491, FM492, FM493	D625	'MULTI-ASSOCIATIONS'
	D626	'TO'
	D627	Member name

Details of ENTITY Members

Format Line Numbers	Parameter Number	Description
FM494, FM495, FM496	D628	'SUB-ENTITIES'
	D629	'ARE'
	D630	Member name

Details of USERVIEW Members

Format Line Numbers	Parameter Number	Description
FM503	D635	'RELATIVE-FREQUENCY'
	D636	Integer
FM506	D637	'RESPONSE-TIME'
	D638	Integer
FM509, FM510, FM511	D639	'DEPENDENCIES'
	D640	'LHS'
	D641	Member name
FM515, FM516, FM517	D642	'FD'/'MVD'
	D643	MVD multiplicity
	D644	'RHS'
	D645	Member name
FM521, FM522, FM523	D646	'DOMAIN'
	D647	'IS'
	D648	Member name
FM527, FM528, FM529	D649	'SUB-DOMAIN'
	D650	'IS'
	D651	Member name

Details of VIEWSET Members

Format Line Numbers	Parameter Number	Description
FM533, FM534, FM535	D652	'CONTAINS'
	D653	Member name

6 Syntax of Commands and Member Types

TRANSLATION-RULE Member Type

This is the full syntax for defining TRANSLATION-RULE members.

```
TRANSLATION-RULE [FOR MANAGER-DICTIONARY member-type]
     [MP-AID-NAME mp-aid-name]
     [common-clauses]
     [VARIABLE Vm ⎧ IS clause-1 [ELSE clause-1]...      ⎫ ]...
                  ⎨ USER-EXIT i PASSING arg [, arg]... ⎬
                  ⎩ translate-clause                   ⎭
CONTENTS
    ⎧ format-line  ⎫
    ⎩ comment-line ⎭
⎧ ; ⎫
⎩ . ⎭
```

where:

member-type is any of the following or a user-defined member type based on any of the following:

ENTITY	IDMS-SUBSCHEMA
FILE	IDMS-VIEW
GROUP	ITEM
IDMS-AREA	MODULE
IDMS-DATABASE	PROGRAM
IDMS-LOGICAL-RECORD	SYSTEM
IDMS-PATH-GROUP	USERVIEW
IDMS-RECORD	VIEWSET
IDMS-SET	

mp-aid-name is the name of the TRANSLATION-RULE member on the MP-AID (if different from the original dictionary member name). It must be no more than 10 characters long.

`common-clauses` are as defined in the *ASG-ControlManager User's Guide*.

`V`*m* is the identifier of a variable, the value of which is defined in the VARIABLE clause. *m* is an integer in the range 0 to 99.

clause-1 is *value*[*clause-2*]

clause-2 is:

$$[\mathtt{IF}\ \left\{\begin{array}{l}\mathit{arg}\\ \mathtt{V}m\end{array}\right\}\left\{\begin{array}{l}\mathtt{EQ}\\ \mathtt{NE}\\ \mathtt{GT}\\ \mathtt{LT}\\ \mathtt{GE}\\ \mathtt{LE}\end{array}\right\}\ \mathit{value}\ [\mathtt{AND}\ \left\{\begin{array}{l}\mathit{arg}\\ \mathtt{V}m\end{array}\right\}\left\{\begin{array}{l}\mathtt{EQ}\\ \mathtt{NE}\\ \mathtt{GT}\\ \mathtt{LT}\\ \mathtt{GE}\\ \mathtt{LE}\end{array}\right\}\ \mathit{value}]\ldots]$$

value is $\left\{\begin{array}{l}\mathtt{D}m\\ \mathtt{U}m\\ \mathtt{S}m\\ \mathtt{'}\mathit{string}\mathtt{'}\\ \mathtt{NULL}\end{array}\right\}$

arg is $\left\{\begin{array}{l}\mathtt{D}m\\ \mathtt{U}m\\ \mathtt{S}m\end{array}\right\}$. A maximum of 15 *arg*s can follow the PASSING keyword.

`D`*m* parameters represent standard attributes and clauses of member definitions.

`U`*m* parameters represent user-defined attributes defined by the User Defined Syntax facility.

`S`*m* refers to a positonal argument passed to this TRANSLATION-RULE from a VARIABLE `V`*m* translate-clause.

string denotes any string of printable characters. A space (hex 40) is considered a printable character.

translate-clause is:

TRANSLATE *arg* USING *mp-tr-rule* [PASSING *arg* [, *arg*]...]

[*clause-2*]

format-line is *format-line-identifier* [*output-specification*]

format-line-identifier is $\mathrm{F}\left\{\begin{array}{l}\mathrm{M}\\ \mathrm{N}\end{array}\right\}\ n[\left\{\begin{array}{l}\mathrm{R}r\\ L\end{array}\right\}\ ]:$

`F` specifies that the line is a format line.

`M` specifies that the output specified by the format line is to be positioned on the next output line.

`N` specifies that the output from this format line is to be treated as a continuation of the previous format line.

`n` is an integer in the range 1 to 9999 that identifies a valid format line number for the corresponding part of the report output.

`R` specifies that the current format line may only be processed a specified number of times.

`r` is an integer in the range 1 to 32767 that specifies the maximum number of times that this format line may be processed.

`L` specifies that the current physical occurrence of information for this format line should be used.

`output-specification` is:

```
[y] ⎧ specifier                             ⎫
    ⎩ (specifier [, specifier]...)          ⎭
[,[y] ⎧ specifier                           ⎫ ]...
      ⎩ (specifier [, specifier]...)        ⎭
```

`y` is an unsigned integer that is a repeat count for the specifier or group of specifier that it immediately precedes.

```
specifier is ⎧ /                                       ⎫
             ⎪ X                                       ⎪
             ⎪ Cc                                      ⎪
             ⎨ [string]                                ⎬
             ⎪ ⎧ Dm ⎫ [.field-width [ ⎧ .R ⎫ ]]        ⎪
             ⎪ ⎨ Um ⎬                 ⎨ .C ⎬           ⎪
             ⎪ ⎪ Vm ⎪                 ⎩ .L ⎭           ⎪
             ⎩ ⎩ Sm ⎭                                  ⎭
```

`/` denotes the end of a physical output line and causes a carriage return/line feed sequence on the output device.

`X` denotes a skip of one character position on the output line.

C*`c`* specifies that the following output is to start at column *`c`*, where *`c`* is an unsigned integer.

`string`, D*m*, U*m*, V*m*, and S*m* are as defined above.

`.field-width` is an unsigned integer immediately preceded by a period (on the same line) specifying the width of the field in which the preceding D*m*, U*m*, V*m*, or S*m* is to be positioned. (The period must appear immediately after the parameter, on the same line. It is not a decimial point; it is a flag indicating to the software that the field width specification follows.) Parameters that have character values are always left justified within a field unless right justification or centering is indicated. Parameters that have integer values are always right justified within a specified field.

`.R` specifies that the immediately preceding parameter is to be right justified within the specified field. If .R is coded, it must appear immediately after the field width specification.

`.C` specifies that the immediately preceding parameter is to be centered within the specified field. If .C is coded, it must appear immediately after the field width specification.

`.L` specifies that the immediately preceding parameter is to be left justified within the specified field. If .L is coded, it must appear immediately after the field width specification.

`comment-line` is `*[string]`

`string` is a string of printable characters comprising a line of comment. A space (hex 40) is considered a printable character. Each comment line must begin with an asterisk (*), which can appear on a line by itself without a specified string of comment.

Remarks

1. If `mp-name` is not specified, and if the dictionary name of the TRANSLATION-RULE member is no more than 10 characters long, the same name will be used for the TRANSLATION-RULE member on the MP-AID. If the dictionary name of the TRANSLATION-RULE member is more than 10 characters long, `mp-name` must be specified or the member cannot be transferred to MP-AID.

2. Comment lines are not included in the output. Each comment line must be contained in one input line, but you can use any number of consecutive comment lines.

3. A maximum of 15 `arg`s can follow the PASSING keyword.

EXTRACT Command

The EXTRACT command imports information and stores it for subsequent translation and/or manipulation. Use this command to extract information from a named source and store it in command variables on the WBTA. This is the command syntax:

```
EXTRACT source
```

where `source` is an external file, a Manager Products command or executive routine, or a command or global variable array.

Information extracted is divided into records and fields. A *record* is either a line of an external file or command output, or an element in the command or global variable array. A *field* is a discrete item of data within a record, separated (by default) by spaces. To specify your own field separator, use the SEPARATOR keyword. Delimited strings within the source are always treated as one field.

Information extracted is divided into records and fields. A record is either a line of an external file or command output, or an element in the Command or Global Variable array. A field is a discrete item of data within a record, separated (by default) by spaces. To specify your own field separator, use the SEPARATOR keyword. Delimited strings within the source are always treated as one field.

By default, each record of the extracted information is stored in a different element of the variable array EXT_RECORD. To store the data in your own variable array(s), use the GENERATE keyword.

You can extract selected parts of your source using a *condition string* if a part of your source meets specific conditions.

You can import empty or blank records (in addition to records which have some content), using the ALL-RECORDS keyword.

You can call specific executive routines with the USING keyword. These routines can manipulate and move the extracted information as required.

Extracting from an External File

To extract information from a sequential file, enter either:

```
EXTRACT EXTERNAL-FILE ddname;
```

or

```
EXTRACT EXTERNAL-FILE ddname SEQUENTIAL;
```

where `ddname` is the name of the job control statement defining the file (up to eight characters long).

To extract information from a member of a partitioned dataset, enter:

```
EXTRACT EXTERNAL-FILE ddname PARTITIONED member-name;
```

where `member-name` is the member of the partitioned dataset.

To extract information from the directory of a partitioned dataset and store the extracted information in your own variable array, enter:

```
EXTRACT EXTERNAL-FILE ddname PARTITIONED DIRECTORY TO store-var;
```

where `store-var` is the name of your variable array.

Alternatively, to additionally extract directory user data, enter:

```
EXTRACT EXTERNAL-FILE ddname PARTITIONED DIRECTORY HEX TO store-var;
```

Note:
By default, empty or blank lines in external files are ignored. To import such records, use the ALL-RECORDS keyword.

You can use EXTRACT EXTERNAL-FILE as the first step in importing objects from an external file into your Manager Products environment.

Extracting from the Output of a Command

To extract information from the output of a command, enter:

```
EXTRACT COMMAND mpr-command;
```

where `mpr-command` is a delimited string giving the name of any primary command (except another EXTRACT command) or any type of executive routine.

Note:
By default, empty or blank lines in a command are ignored. To import such records, use the ALL-RECORDS keyword.

Extracting from a Variable Array

To extract information from a command or global variable array, enter:

```
EXTRACT VARIABLE input-var ALL;
```

where `input-var` is the name of the variable array.

To copy information extracted from a variable array to your own variable array, excluding null elements and duplicate elements, enter:

```
EXTRACT VARIABLE input-var DISTINCT TO store-var;
```

where `store-var` is the name of your own variable array.

Note:
By default, empty or blank lines in a command are ignored. To import such records, use the ALL-RECORDS keyword.

Storing Extracted Data in Specified Variable Arrays

To store extracted data in your own command variable arrays, enter:

```
EXTRACT source GENERATE variables;
```

where `variables` give rules specifying which variables will hold the extracted information, to be applied repeatedly to the records of the extracted data. They are defined by the positions of the variable names in relation to each other, and special characters. These are the special characters:

- A period (.) skips over a field.
- A comma (,) skips to the start of the next record.
- A percent (%) stores a whole record in a variable, whose name must immediately follow this character, and must appear between two commas or be the first or last element of the rules specified.

For example, to extract information from an external file CMPINF, and store it in the two variable arrays NAME and CONTENTS, enter:

```
EXTRACT EXTERNAL-FILE CMPINF GENERATE 'NAME,%CONTENTS'
```

NAME(1) will hold the first field of the first input line. CONTENTS(1) will then hold the whole of the next input line. The process will then continue, for NAME(2), CONTENTS(2), and repeat until extraction stops.

Specifying a Field Separator

To override the default field separation character (space), enter:

```
EXTRACT source SEPARATOR separator;
```

where *separator* is a single (delimited) character.

For example, if you are extracting from an external file with ddname DATA3, which has a comma as its field separator, you can override the default field separator for that file, by entering:

```
EXTRACT EXTERNAL-FILE DATA3 SEPARATOR ',';
```

Storing Extracted Data in Specified Parts of Variables

To specify an array element number for your storage variable(s), at which to start storing extracted information, enter:

```
EXTRACT source ARRAYNUM num;
```

where *num* is the array element number.

For example, to extract the contents of an external file with a job control definition of CFILE1, starting storing the lines of that file in the tenth array element (of EXT_RECORD), enter:

```
EXTRACT EXTERNAL-FILE CFILE1 ARRAYNUM 10;
```

Note:

You can store the results of multiple extractions in one variable array (or set of arrays) by storing extracted information in different parts of the array.

Extracting Selected Data

You can specify conditions under which data is extracted using a condition string (and optionally a column position) by entering either:

```
EXTRACT source keyword condition;
```

or

```
EXTRACT source keyword POS position condition;
```

where:

condition is a delimited string matched against each record in your source. If a match occurs, the appropriate action (specified by keyword) is taken. By default, the condition string is matched against characters starting from the first character in each record (ignoring leading blanks).

Note:

You can specify several conditions for starting and ending extraction, but since extraction can only start and end once, no more than one start and end condition (the first encountered) can be met.

position is a number specifying a column position, to match the string against characters starting in that position in each record.

keyword is any of the keywords listed in the table below, along with the action taken if a match occurs. You can use combinations of keywords in one command, to suit your requirements. Each keyword can be repeated up to 180 times.

Some keywords deal with groups. A group is the amount of data needed to populate one element of each variable defined in the GENERATE clause (by default, this is one element of EXT_RECORD). Groups always start on a new line of input.

Keyword	Action
STARTF or STARTA	Start extracting data from or after the first record on which a match is found.
ENDF or ENDA	End extracting data from or after the first record on which a match is found.
SELECTG or SKIPG	Select or skip over groups of data. Selection or skipping stops at the end of the group. Only groups whose first record contains the string are selected.
SELECTR or SKIPR	Select or skip over those records containing the condition string.

Keyword	Action
SELECTA or SKIPA	Select or skip over records which have the condition string occurring anywhere in the record. You cannot specify a column position with the SELECTA or SKIPA keywords.
SYNC	If this string is found when processing a group, then start storing data in a new group.
SKIP	Skip all records in a group until the condition string is matched. When this occurs, start storing data in a new group.

You can also extract selected data based on a definite range within your source. For example, to select the first 100 lines in a file enter:

```
EXTRACT source RECORDS rec1;
```

or

```
EXTRACT source RECORDS rec1 TO rec2;
```

where:

rec1 is the record number at which data storage should start.

rec2 is the record number after which data storage should end. *rec2* defaults to the end of data in your source.

To apply extraction to an inclusive range of columns only, enter:

```
EXTRACT source COLUMNS col1 TO col2;
```

where *col1* and *col2* define the start and end column positions. Data outside these positions is ignored. The resulting truncated records are treated as whole records, with *col1* becoming the first column of these new records.

The above keywords are processed in a set order.

First, any start conditions must be satisfied in the sequence:

- STARTA
- STARTF
- RECORDS

after which each record is processed according to this sequence:

- COLUMNS
- ENDA
- ENDF
- RECORDS
- SELECTG
- SYNC
- SKIPS
- SELECTR
- SKIPR
- SKIPG
- SELECTA
- SKIPA

Null Filling Elements of the Output Array

To null fill elements in the output array corresponding to unselected elements in the input data, enter:

```
EXTRACT source keyword condition NULL-FILL;
```

The effect of NULL-FILL is illustrated in these diagrams.

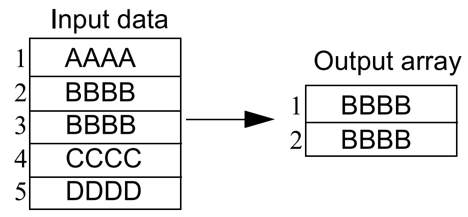


Figure 6. EXTRACT selecting BBBB without NULL-FILL

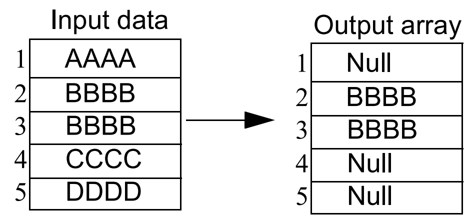


Figure 7. EXTRACT selecting BBBB NULL-FILL

NULL-FILL can be truncated to N.

Importing Empty or Blank Records

To import empty or blank records (in addition to records which have some content), enter:

```
EXTRACT source ALL-RECORDS;
```

One cell of the variable created is space-filled for every blank record imported. The number of spaces in the cell are specified this way:

- If source is an external file, the logical record length of the external dataset from which the records are being imported
- If source is a command or global variable array, the length of the input variable
- If source is a Manager Products command or executive routine, 1

If you are using the GENERATE keyword to extract data selectively, a variable type of null is created to indicate the presence of a blank record with no identifiable elements.

Using an Executive Routine

To specify an executive routine to be used, enter:

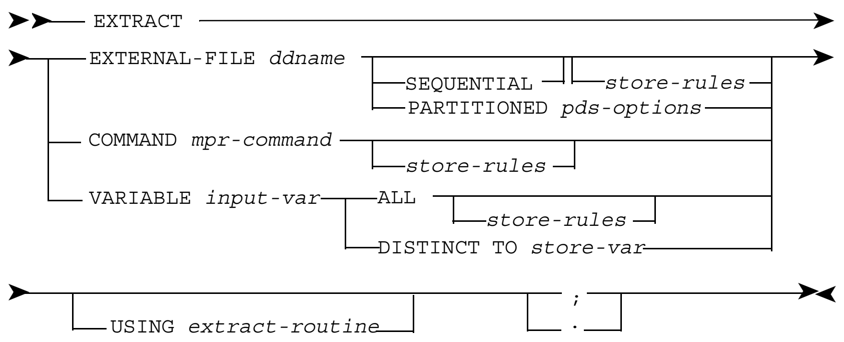
```
EXTRACT source USING extract-routine;
```

where *extract-routine* is an executive routine you supply to manipulate extracted data. You can only call one routine, but this can invoke other executive routines.

If you intend to populate your repository with members generated from the extracted data, the executive routine(s) you use must ensure that specific elements of the extracted information are stored in mandatory variables, to be used during the reconcile and preview stages of the import process.

Refer to Chapter 4, "Import Executive Routines and Variables," on page 43 for further details of executive routines.

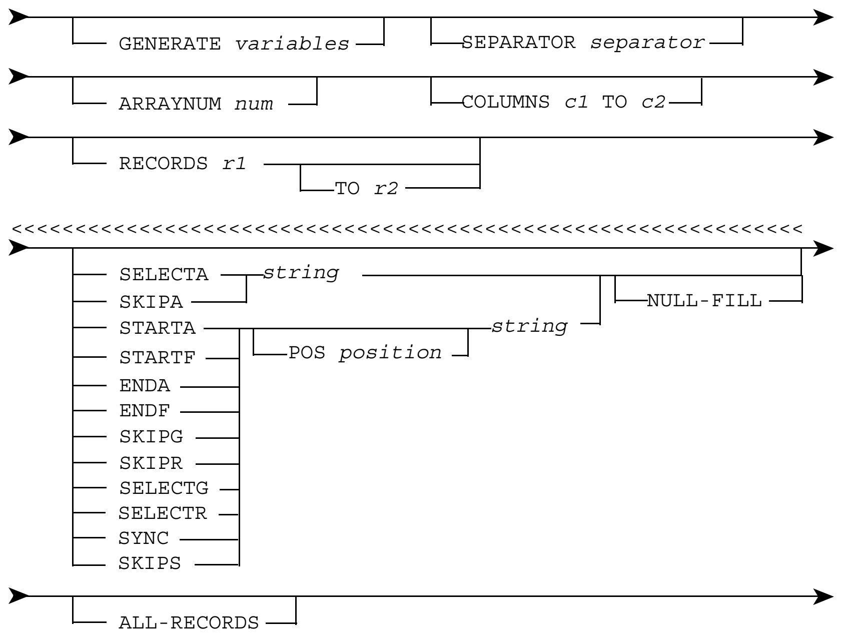
Syntax



where:

ddname is the name of an external file or dataset.

`store-rules` may be repeated up to 180 times. This is the syntax:



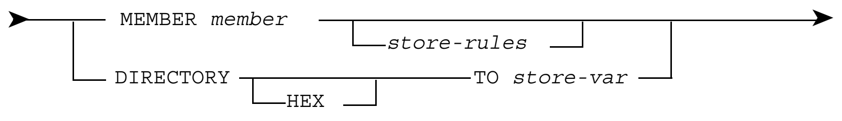
`variables` is a delimited string that specifies how extracted data is stored.

`separator` is a single delimited character that specifies a field separator.

`num`, `c1`, `c2`, `r1`, `r2`, and `position` are integers.

`string` is a delimited string that specifies a condition.

`pds-options` are:



`member` is the name of a partitioned dataset member.

`store-var` is the name of the variable into which extracted data is stored.

`store-rules` are as defined above.

`mpr-command` is a delimited string that gives any executive routine or Manager Products primary command (except EXTRACT).

`input-var` is the name of the variable from which input is to be taken.

`extract-routine` is the name of an executive routine.

POPULATE Command

The POPULATE command populates the repository with the results of the preceding PREVIEW IMPORT command. Use POPULATE to execute the ADD or REPLACE command and member definition statements generated by a previous PREVIEW IMPORT command. The statements are those displayed in the current buffer or filed in a USER-MEMBER.

To execute statements displayed in the current buffer, enter:

```
POPULATE FROM BUFFER;
```

The output of the PREVIEW command must be displayed in the current buffer which can be a Command Mode, Lookaside, Update, or Edit Buffer.

If you want to enter other commands between a PREVIEW and POPULATE command, you can prefix these other commands with BROWSE or LOOKASIDE and then use QUIT or XQUIT to return to the output of the PREVIEW command before entering POPULATE.

To execute ADD and REPLACE statements filed in a USER-MEMBER enter:

```
POPULATE FROM USER user-member-name;
```

where *user-member-name* is the name of the USER-MEMBER in which the command and member definition statements generated by a previous PREVIEW IMPORT command have been filed.

You can also execute the command and member definition statements by editing the USER-MEMBER and entering a RUN command or by entering the name of the USER-MEMBER in the command area.

By prefixing POPULATE with a NOPRINT command you can prevent output from being printed.

Refer to the *ASG-ControlManager User's Guide* for details of the commands mentioned above.

Specifying that Statements will form a Logical Unit of Work

You can specify that the command and member definition statements entered in the repository by the POPULATE command are to be treated as one logical unit of work (LUW).

To specify that all the statements will form one LUW, enter:

```
POPULATE FROM BUFFER ROLLBACK;
```

or

```
POPULATE FROM USER user-member-name ROLLBACK;
```

By using the ROLLBACK keyword, you can specify that all the statements will form a LUW that will either update the repository or be rolled back in its entirety, leaving the repository unchanged, if for any reason any of the statements are unsuccessful.

For example, you can avoid a situation where the definition of the parent object is entered in the repository but the definitions of some of its children are not.

To change the message error level which causes ROLLBACK to occur, enter:

```
POPULATE FROM BUFFER ROLLBACK LEVEL error-level;
```

or

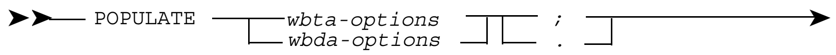
```
POPULATE FROM USER user-member-name ROLLBACK LEVEL error-level;
```

where `error-level` can be:

- E, to cause ROLLBACK when error messages occur
- W, to cause ROLLBACK when warning messages and error messages occur

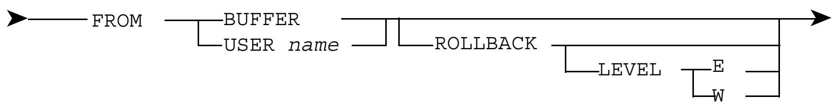
If you do not use the LEVEL keyword, E is the default (so ROLLBACK is not normally caused when warning messages occur).

Syntax



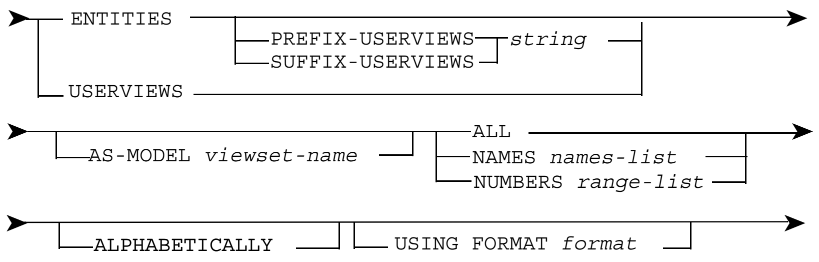
where:

wbta-options are:



name is the name of a USER-MEMBER on the MP-AID.

wbda-options are:



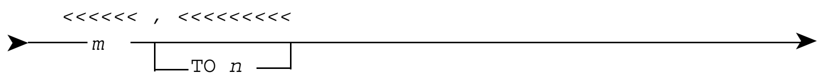
`string` and `viewset-name` can contain from 1 to 32 characters and must conform to the Manager Product rules for valid member names.

Note:

If a supplementary USERVIEW member is required for an ENTITY being defined, `string` is prefixed or suffixed with the name of the ENTITY to produce the name of the supplementary USERVIEW. If the resulting name contains more than 32 characters it is reduced.

`name-list` is a list of names of relations (USERVIEWS) or records (ENTITIES) separated by commas.

`range-list` is:



m and *n* are numbers assigned in the WBDA to relations (USERVIEWS) or records (ENTITIES). If present, *n* must be greater than *m*.

`format` is the name of a FORMAT member of the modeling repository used by user-formatted output functions.

`wbta-options` are provided by generic import functions.

`wbda-options` are provided by data modeling and design functions.

PREVIEW IMPORT Command

The PREVIEW IMPORT command uses the information on the WBTA—as it has been processed by previous RECONCILE, RADD, RIGN, RREN, RREP, or RUPD commands—to generate ADD or REPLACE command- and member-definition statements.

To generate the command- and member-definition statements, enter:

```
PREVIEW IMPORT;
```

The member definition statements are generated in the default layouts provided by Manager Products for each member type. You can create layout rules that suit your repository standards and invoke them in the USING keyword of the PREVIEW IMPORT command.

The NOTE attribute contains the date and time that the member definition statement was generated by the PREVIEW command. The ALIAS attribute contains the external object’s name. The alias type will correspond to the language used in the external environment from which information about the object was imported.

Information in the NOTE and ALIAS attributes of the existing member is incorporated in those of the proposed member. The single ALIAS attribute generated could contain different aliases of the same alias type. You must edit the member definition statement generated by the PREVIEW command and change one of the aliases.

If the definition is to replace an existing member then certain default common attributes of the existing member are incorporated in the definition, unless you have specified the NO-COMMON-CLAUSES keyword in a previous RECONCILE command.

PREVIEW IMPORT processes members on the WBTA in the same order as they are listed on the Reconciliation Report generated by the previous RECONCILE command. A proposed member can appear more than once in a Reconciliation Report, but the PREVIEW IMPORT command only generates one command and member definition statement for each member.

For example, you could import information about two tables which share a column of the same name. The shared column would appear twice on the reconciliation report but only one command and member definition statement would be generated to document it in the repository.

A member whose definition is not generated, because:

- It has been ignored by a previous RECONCILE command
- It has already been generated in the current PREVIEW IMPORT output

is indicated by comments. These comments help you to relate the PREVIEW output with the previous reconciliation report.

The generated statements can be printed, filed in a USER-MEMBER on the MP-AID, or both printed and filed.

To file the generated statements in a USER-MEMBER you must specify an ONTO keyword in the PREVIEW IMPORT command.

By filing the command- and member-definition statements in a USER-MEMBER, you can hold them across Manager Products sessions and edit the generated statements in the edit buffer.

You can subsequently enter the statements in the repository using the POPULATE command.

Generating Member Definition Statements in Your Own Layouts

You can tailor the PREVIEW IMPORT command so that it generates member definition statements in layouts which suit your repository standards.

To generate member definition statements in your own layouts, enter:

```
PREVIEW IMPORT USING layout-executive;
```

where `layout-executive` is an executive routine that invokes other executive routines that each determine the layout of member definition statements generated for a particular member type.

Alternatively, you can tailor the executive routines in Manager Product’s default layout rules.

Filing Generated Output in a USER-MEMBER

To automatically file generated output in private or public USER-MEMBERs on the MP-AID, enter:

`PREVIEW IMPORT ONTO` *`member-type member-name options;`*

where:

`member-type` is the type of USER-MEMBER which will hold the generated output: USER-MEMBER, PUBLIC-USER-MEMBER, or PRIVATE-USER-MEMBER, to file output in a public or private USER-MEMBER (USER-MEMBER is private).

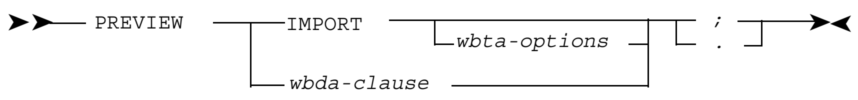
When appending or replacing the contents of an existing member the user who created that member can change it from private to public, or the reverse, by specifying either PRIVATE or PUBLIC. Users with different logon identifiers can create private USER-MEMBERs with the same names.

`member-name` is the name of the USER-MEMBER.

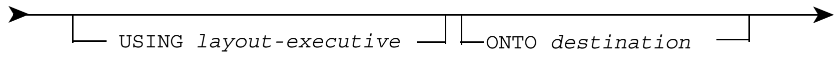
`options` define how the output is to be filed. These are the keywords:

- — NEW (the default), APPEND, or REPLACE, to create a new member, append to an existing member, or replace an existing member. If you specify NEW, and the member already exists, the output will not be generated. If you specify APPEND or REPLACE, and the member does not already exist, a new member is created.
- — PRINT (the default), to print the full output, or NOPRINT, to print messages and impact analysis reports only.

Syntax

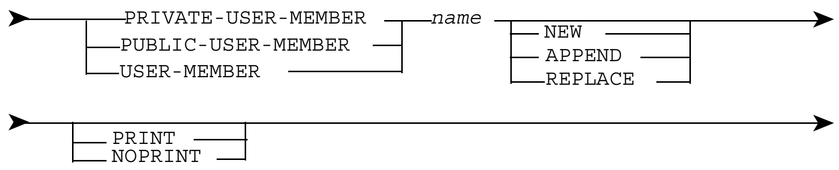


where *`wbta-options`* are:



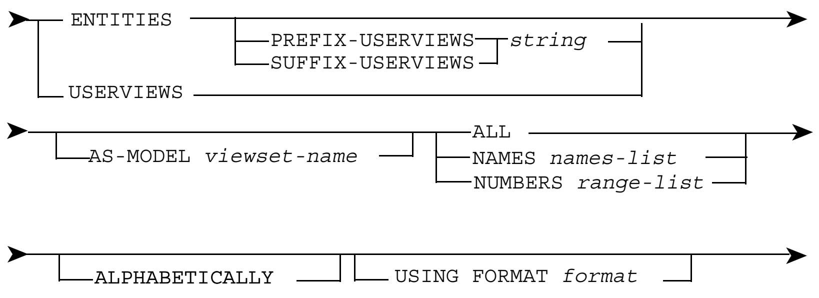
`layout-executive` is the name of an executive routine.

`destination` is:



`name` is the name of a USER-MEMBER on the MP-AID.

wbda-clause is:



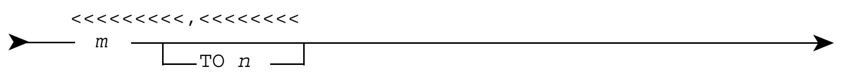
string, *viewset-name* can contain from 1 to 32 alphanumeric characters and must conform to the Manager Product rules for valid member names.

Note:

If a supplementary USERVIEW member is required for an ENTITY being defined, *string* is concatenated (either as a prefix or a suffix) with the name of the ENTITY to produce the name of the supplementary USERVIEW. If this resulting name contains more than 32 characters, it is truncated.

name-list is a list of names of relations (USERVIEWS) or records (ENTITIES) separated by commas.

range-list is:



where *m* and *n* are numbers assigned in the WBDA to relations (USERVIEWS) or records (ENTITIES). If present, *n* must be greater than *m*.

format is the name of a FORMAT member of the modeling repository. FORMAT members are used by User Formatted Output functions.

Note:

The PREVIEW IMPORT command is provided by generic import functions. The *wbda-clause* is provided by data modeling and design functions.

RADD

Use RADD to specify that you want a proposed member documenting an external object to be added to the repository.

To use the RADD line command, enter:

```
RADD
```

in the line command area alongside the proposed member's identification number in the reconciliation report.

To use the RADD primary command, enter:

```
RADD n;
```

in the command area.

where *n* is the proposed member's identification number in the reconciliation report.

The reconciliation report is displayed by the RECONCILE command. To display the changes you have made with the RADD command enter a further RECONCILE command.

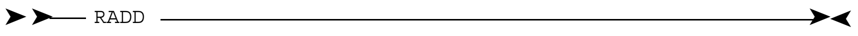
The effects of the two forms of the RADD command are the same but you can only enter line commands when working in an interactive environment. You can enter several RADD line commands at the same time.

If a member with the same name as a proposed member already exists in the repository and you specify RADD, it will be taken to mean replace.

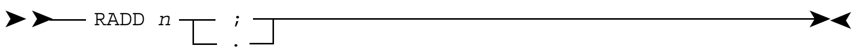
You can also specify that you want a proposed member to be added to the repository by including an ADDING keyword in the RECONCILE command.

Refer to the RECONCILE command for full details of adding proposed members.

Syntax (Line Command)



Syntax (Primary Command)



where *n* is a proposed member's identification number in a reconciliation report.

RECONCILE Command

The RECONCILE command uses translation rules to generate proposed members from the information about external objects placed on the WBTA by the EXTRACT command.

You can override the translation rules by specifying an ADDING, IGNORING, NO-COMMON-CLAUSES, RENAMING, or REPLACING keyword in a RECONCILE command.

You can also tailor the Manager Products default translation rules or create your own translation rules and invoke them in the USING keyword of a RECONCILE command.

A member name and member type are generated for the proposed members. A form-description and (depending on the data type of the column it is documenting) a USAGE attribute are generated for proposed members which have an ITEM member type.

The RECONCILE command does not update the repository, but rather specifies the updates you intend to make. These updates are determined by the current contents of the repository. A proposed member will be added to the repository unless a member (other than a dummy member without a source record) of the same name already exists.

Existing ITEM members will be replaced by proposed members whose definitions contain additional form descriptions. The form descriptions of the existing member are included in the definition of the proposed member.

In all other cases, if a proposed member has the same name as an existing member, it will not be updated.

The RECONCILE command can be entered any number of times. Any Manager Products instruction other than LOGOFF or RELEASE GLOBAL can be entered between RECONCILE commands. If you specify an EXTRACT command between RECONCILE commands then the information on the WBTA is replaced and subsequent RECONCILE commands will apply to the latest and not the previously imported information.

The first RECONCILE command generates the proposed members. Subsequent RECONCILE commands can both change the proposed members and, subject to member type checks, specify whether or not they are to be entered in the repository. You can also regenerate the proposed members by entering a RECONCILE command including an INITIALIZE keyword. The proposed members are regenerated according to the translation rules and any changes you have made to the proposed members previously generated are abandoned. A Reconciliation Report is displayed by each RECONCILE command. The report compares the proposed members with existing repository members which have the same name.

You can use the reconciliation report to reconcile the proposed and existing repository members with one another. You can also use the RADD, RIGN, RREN, RREP, and RUPD commands during reconciliation. In an interactive environment, these can be line commands or primary commands.

The reconciliation report displays the condition of the existing members at the time the proposed members were generated or regenerated. Reconciliation reports displayed in response to subsequent RECONCILE commands not including an INITIALIZE keyword will display any changes you have made to the proposed members but will not display any changes in the condition of the existing members.

The Systems Administrator can define member type checks that specify the types of existing members to which all the proposed members documenting a parent object and its children can refer. A check can fail, partially fail, or pass.

If the check fails, the proposed member and all other members in the parent-children set cannot be added to the repository. This condition is indicated by an error message in the reconciliation report and you cannot override it with an RADD, RECONCILE ADDING, RREP, or RECONCILE REPLACING command.

If the check partially fails, the proposed member can be added to the repository but a warning message in the reconciliation report indicates the partial failure.

If the check passes, then the proposed members can be added to the repository as normal.

For example, your Systems Administrator could specify that columns in tables should preferably be documented in ITEM members but can be documented in GROUPs. This check will fail if an existing SYSTEM member has the same name as a proposed member documenting a column but will only partially fail if the existing member is a GROUP.

Member type checking enables you to take early action to ensure that proposed members will not fail to encode, due to a reference to an existing member with an invalid member type, when the repository is populated.

When a member type check failure occurs, you can rename the offending proposed member to a name that does not exist in the repository or to an existing member name that does not cause a further check failure. A RECONCILE RENAMING command rechecks all the proposed members in the parent-children set.

The updates to be made to the repository are determined by the contents of the current or next visible status. The reconciliation report displays the condition of the existing members in the current or next visible status. If you change statuses, the report will continue to display the members as they are visible from the previous status unless you regenerate the proposed members by specifying an INITIALIZE keyword.

Regenerating Proposed Members

To regenerate the proposed members documenting external objects, enter:

```
RECONCILE INITIALIZE;
```

The proposed members are regenerated according to the default translation rules. Any changes you have made to the previously generated members (for example renaming them) are abandoned. The reconciliation report will display the current condition of any repository member whose name is the same as a proposed member.

Tailoring How Proposed Members are Generated

To tailor the RECONCILE command so that it generates proposed members which suit your repository standards, enter:

```
RECONCILE INITIALIZE USING translation-rule-name-list;
```

where *`translation-rule-name-list`* is a list of one or more executive routines each separated by a comma. The executive routines must be listed in the order they are to be executed.

For example, if Manager Product’s default naming rules for proposed members do not suit your repository standards you can create executive routines specifying alternative rules. Alternatively, you can tailor the executive routines in the Manager Products default translation rules.

Stopping Proposed Members being Entered in the Repository

To ignore a selection of proposed members so that they are not subsequently entered into the repository, enter:

```
RECONCILE IGNORING selection;
```

where `selection` specifies which of the proposed members are to be ignored. Refer to "Selecting Members to be Ignored, Added, or Replaced" on page 90 for details of selection.

The reconciliation report will indicate that a proposed member is to be ignored.

You can also use the RIGN command to ignore proposed members.

You cannot ignore proposed members documenting referenced objects.

Adding Proposed Members

To specify that you want a selection of proposed members to be added to the repository (and not ignored), enter:

```
RECONCILE ADDING selection;
```

where `selection` specifies which of the proposed members you want to be added to the repository. Refer to "Selecting Members to be Ignored, Added, or Replaced" on page 90 for details of selection.

Existing members with the same name as proposed members will be replaced as a result of a RECONCILE ADDING command.

You can also specify that a proposed member will replace an existing member by entering a RECONCILE REPLACING or RREP command.

You can rename proposed members by entering a RECONCILE RENAMING or RREN command.

If member type checking has been enabled by your Systems Administrator and you want to replace an existing member with a proposed member, then the member type of the existing member is checked against a set of allowed proposed member types. If the check fails, the command will not be executed.

The reconciliation report will indicate that the proposed members will be added to the repository as a new member or replace an existing member.

You can also use the RADD command to specify which proposed members are to be added to the repository.

You cannot specify that proposed members documenting referenced objects are to be added to the repository. Referenced objects are added to the repository as dummy members by a reference from the member documenting the parent object, if these members are not already present on the repository.

Replacing Existing Members with Proposed Members

To specify that you want a selection of proposed members to replace existing repository members, enter:

```
RECONCILE REPLACING selection;
```

where `selection` specifies which proposed members are to replace existing members. Refer to "Selecting Members to be Ignored, Added, or Replaced" on page 90 for further details of selection.

The reconciliation report will indicate which of the proposed members are to replace existing repository members.

You can also use the RREP command to specify which proposed members are to replace existing members.

If member type checking is enabled by your Systems Administrator, then the member type of the existing member being replaced is checked against a set of allowed proposed member types. If the check fails, the command is not executed.

You cannot specify that proposed members documenting referenced objects are to replace existing repository members. A relationship is created in the repository between the proposed member documenting the parent object and the existing member.

Renaming Proposed Members

To rename a selection of proposed members, enter either:

```
RECONCILE RENAMING MEMBER member-name-1 AS member-name-2;
```

or

```
RECONCILE RENAMING NUMBER n AS member-name-2;
```

where:

`member-name-1` is the current name of the proposed member.

`member-name-2` is the new name.

`n` is the proposed member's identification number in the reconciliation report.

You can rename several proposed members with one RECONCILE command by repeating the MEMBER and NUMBER keywords. The reconciliation report will display the new member names of the proposed members.

If a proposed member appears more than once in the same reconciliation report then a RECONCILE command including a MEMBER keyword will rename all occurrences of the proposed members in the report.

If `member-name-2` is the same as the name of another proposed member in the reconciliation report, then the command will be rejected. If it is the same as the name of an existing member, then the existing member is displayed in the reconciliation detailed report.

If you rename children, then the proposed member documenting the parent object will refer to the children by their new names.

You may want to rename a proposed member if:

- It has the same name as an existing member
- Its name does not suit your naming standards
- It has failed or partially failed member type checking

You can create your own naming rules and invoke them in the USING keyword of a RECONCILE command or tailor the executive routines in the Manager Products supplied naming rules. You can also use the RREN command to rename proposed members.

Selecting Members to be Ignored, Added, or Replaced

You can select which proposed members you want to be ignored, added to the repository or replace existing repository members.

To select proposed members by their member type, enter:

```
RECONCILE update TYPE member-type-list;
```

where:

`update` is IGNORING, REPLACING, or ADDING.

`member-type-list` is a list of member types each separated by a comma.

To select proposed members by their member name, enter:

```
RECONCILE update MEMBER member-name-list;
```

where *`member-name-list`* is a list of member names each separated by a comma.

To select proposed members by their identification number in the reconciliation report, enter:

```
RECONCILE update NUMBER id-number-list;
```

where *`id-number-list`* is a list of identification numbers each separated by a comma.

To select all proposed members, enter:

```
RECONCILE update GROUP ALL;
```

To select those proposed members which have the same name as an existing repository member, enter:

```
RECONCILE update GROUP DUPLICATES;
```

The different updates and selections can be combined in a single RECONCILE command. For example, to specify that:

- Proposed members will replace existing repository members which have the same member name
- Proposed members with a member type of MODULE and the proposed member with the member name IT-DEPT-NAME will not be entered in the repository

enter:

```
RECONCILE REPLACING GROUP DUPLICATES IGNORING TYPE MODULE MEMBER
IT-DEPT-NAME;
```

Excluding Common Clauses from the Definition of Proposed Members

To stop the default common attributes of existing repository members being incorporated in proposed members which are replacing them, enter:

```
RECONCILE NO-COMMON-CLAUSES;
```

You can specify with a RECONCILE REPLACING or RREP command that a proposed member is to replace an existing repository member.

If you do not enter a RECONCILE command including the NO-COMMON-CLAUSES keyword then the ADMINISTRATIVE-DATA, ALIAS, COMMENT, DESCRIPTION, and NOTE default common attributes of the existing repository member are incorporated in the definition of the proposed member replacing it.

Proposed members always have a NOTE and an ALIAS attribute. The attributes are displayed in the member definition statements generated for the proposed members by a subsequent PREVIEW command. The NOTE attribute gives the time and date the statement was generated. The ALIAS attribute gives the name of the external object the definition is documenting.

If the default common attributes of the existing member are updated after the proposed members were last generated by the RECONCILE command then the updates are not reflected in the member definition statements generated by the PREVIEW command.

Specifying the Type of Reconciliation Report you want Displayed

You can specify the type of reconciliation report you want to be displayed by a RECONCILE command.

To display a summarized reconciliation report, enter:

```
RECONCILE LIST SUMMARY;
```

To display a detailed reconciliation report, enter:

```
RECONCILE LIST DETAILS;
```

To display both a detailed and a summarized reconciliation report, enter:

```
RECONCILE LIST BOTH;
```

The summarized report is displayed by default.

To display a detailed reconciliation report excluding certain information about the relationships between objects, enter:

```
RECONCILE LIST DETAILS NO-XREF;
```

If you specify the NO-XREF keyword then these are not displayed in the detailed reconciliation report:

- The table listing the children of the parent object
- The Also Referred To By Entry which indicates that children are shared by more than one of the parent objects on which information has been imported

A Description of the Reconciliation Summary Report

A reconciliation summary report lists the proposed members documenting the external objects about which information has been imported.

The ID column contains the unique identification number of the proposed members. The identification number specifies the order in which information about each object was imported. This order is determined by the object’s type. The numbering in the report is not in sequence if there is more than one parent object because information on several objects of the same type is imported. Parent objects have the lowest identification numbers and are followed in the report by their children.

The Proposed Member Name column contains the name of the proposed members.

The Type column contains the member type of the proposed members.

The Upd column specifies how the repository is to be updated with the proposed members. If ADD is specified, the proposed member is to be added to the repository. If REP is specified the proposed member is to replace an existing member in the repository. If IGN is specified the proposed member is not to be entered in the repository. An asterisk (*) indicates that the proposed member is a referenced object and is added to the repository as a dummy member by a reference from the member documenting the parent object (if the member does not already exist).

Initially:

- IGN is specified if there is an existing member with the same name as the proposed member.
- REP is specified if the proposed member is an ITEM member with a form description not included in the existing member.
- ADD is specified if there is no existing member.

The Condition column shows whether there is an existing repository member with the same name as the proposed member and if it is a dummy, encoded, unverified, or protected member. If the column is blank no member of the same name exists. `*NO AUTH` is displayed if you do not have the authority to access the existing member. The entries in the Condition column are otherwise the same as those displayed in the Condition column of LIST command output.

If the Systems Administrator has enabled member type checking and a proposed member fails the check, the error message `DM05784E` is displayed. If a proposed member partially fails the check, the warning message `DM05784W` is displayed.

Refer to the *ASG-Manager Products Dictionary/Repository User's Guide* for details of the LIST command.

An Example of the Reconciliation Summary Report

This is an example reconciliation summary report:

```
***************************************************************************
Reconciliation summary report for extract of table AAW.SALES from DB2.
***************************************************************************
ID        Proposed Member name        Type                Upd      Condition
---------------------------------------------------------------------------
1         TB-AAW-SALES                DB2-TABLE           ADD
2         US-AAW                      DB2-USER            IGN       SCE ENC
3         TS-NORTH                    DB2-TBSPACE         *
4         IT-QTY                      ITEM                ADD       *   DUM
5         IT-DESCRIPTION              ITEM                IGN       SCE ENC
6         IT-DELIVERY                 ITEM                ADD
7         IT-PRICE                    ITEM                REP       SCE ENC
---------------------------------------------------------------------------
```

A Description of the Reconciliation Detailed Report

A reconciliation detailed report is divided into different sections for each external object about which information has been imported. Each object has a unique identification number which specifies the order in which information about it was imported. This number is the same as the number in the summary report.

The entries following Extracted give the name and type of the external object.

The entries following Refers to give the name and type of a referenced object.

The entries following Proposed give the member name and member type of the proposed member documenting the external object and indicate whether the member is to be added to the repository, replace an existing dictionary member, or be ignored. Proposed members documenting referenced objects are indicated by an asterisk (*).

If a proposed member is an ITEM, then its form description (depending on the data type of the column it defines), USAGE attribute, and version are also displayed.

The entries following Dictionary give information about the condition (encoded, dummy, or unverified) and member type of any repository member with the same name as the proposed member. If the existing member is an ITEM then all versions of its form-description and their associated USAGE attributes are displayed. `*NO AUTH` is displayed if you do not have the authority to access the existing member.

The section reporting the parent object is followed by a list of the children (including referenced objects) on which information has been imported.

If information has been imported from more than one parent object and they share the same children this is indicated by an Also Referred To By Entry in the sections reporting the shared children.

If the Systems Administrator has enabled member type checking and a proposed member fails the check, the error message `DM05784E` is displayed. If a proposed member partially fails the check the warning message `DM05784W` is displayed.

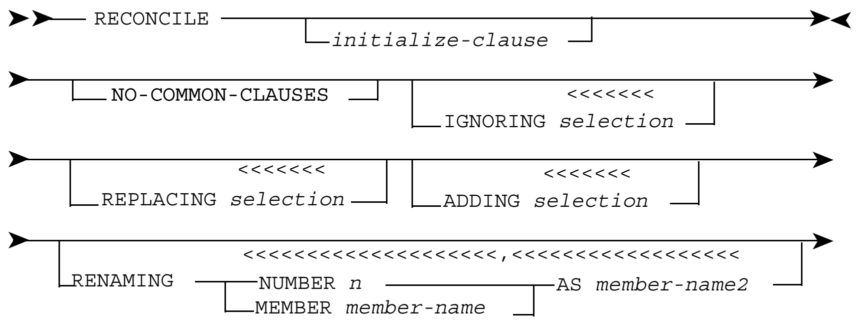
An Example of the Reconciliation Detailed Report

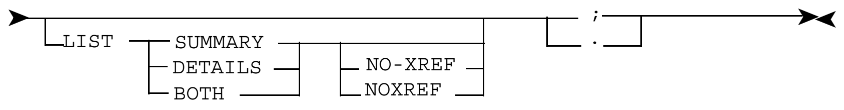
This is an example reconciliation detailed report:

```
*****************************************************************
Reconciliation detailed report for extract of table AAW.SALES from DB2.
******************************************************************

 1   Extracted... SALES                               TABLE
     Proposed.... TB-AAW-SALES     DB2-TABLEADD
-----------------------------------------------------------------
------------- 6 CHILDREN extracted with AAW.SALES ---------------
-----------------------------------------------------------------
              1  CREATOR
              1  DBSPACE
              4  COLUMNS
-----------------------------------------------------------------
2   Extracted... AAW                                  CREATOR
    Proposed.... US-AAW            DB2-USERGIGN
    Dictionary.. SCE ENC           DB2-USER
-----------------------------------------------------------------
3   Refers to... NORTH                                TABLESPACE
    Proposed.... TS-NORTH                  DB2-TBSPACE*
-----------------------------------------------------------------
4   Extracted... QTY                                  COLUMN
    Proposed.... IT-QTY                    ITEM       ADD
    Dictionary.. *   DUM           ITEM
-----------------------------------------------------------------
5   Extracted... DESCRIPTION                      COLUMN
    Proposed.... IT-DESCRIPTION    ITEM           IGN
    Form desc.. CHARACTERS 5       VERSION 1
    Dictionary.. SCE ENC           ITEM
    Version 1.. HELD-AS CHARACTERS 5
    Version 2.. ENTERED-AS CHARACTERS 4
-----------------------------------------------------------------
6   Extracted... COST                                 COLUMN
    Proposed.... IT-DELIVERY       ITEM           ADD
    Form desc.. CHARACTERS 10 USAGE DATEVERSION 1
-----------------------------------------------------------------
7   Extracted... PRICE                                COLUMN
    Proposed.... IT-PRICE                  ITEM       REP
    Form desc.. NUMERIC 6                  VERSION 3
    Dictionary.. SCE ENC           ITEM
    Version 1.. HELD-AS CHARACTERS 5
    Version 2.. ENTERED-AS CHARACTERS 4
*****************************************************************
```

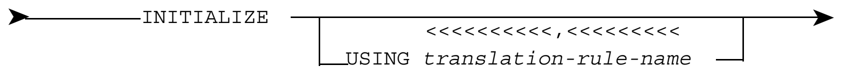
Syntax





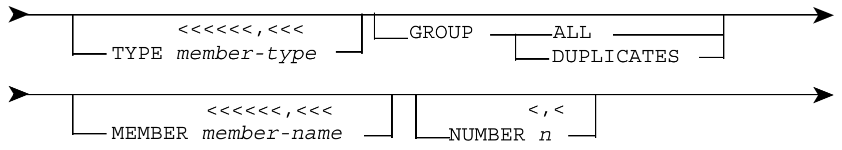
where:

initialize-clause is:



translation-rule-name is the name of an executive routine.

selection-clause is:



member-type is the member type of a proposed member.

member-name is the name of a proposed member.

n is a proposed member's identification number in a reconciliation report.

RIGN

Use RIGN to specify that you do not want a proposed member documenting an external object to be entered in the repository.

To use the RIGN Line Command, enter:

```
RIGN
```

in the line command area alongside the proposed member's identification number in the reconciliation report.

To use the RIGN primary command, enter:

```
RIGN n;
```

in the command area.

where *n* is an integer identifying the proposed member's identification number in the reconciliation report.

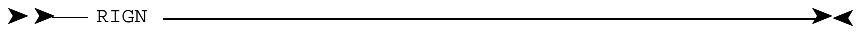
The effects of the two forms of the RIGN command are the same but you can only enter line commands when working in an interactive environment. You can enter several RIGN line commands at the same time.

The reconciliation report is displayed by the RECONCILE command. To display the changes you have made with the RIGN command enter a further RECONCILE command.

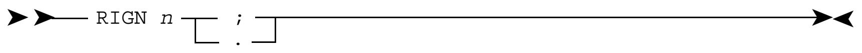
You can also specify that you want a proposed member to be ignored by including an IGNORING keyword in the RECONCILE command.

Refer to the RECONCILE command for full details of ignoring proposed members.

Syntax (Line Command)



Syntax (Primary Command)



where *n* is a proposed member's identification number in a reconciliation report.

RREN

RREN renames a proposed member during reconciliation.

To use the RREN line command, enter:

```
RREN
```

in the line command area alongside the proposed member's identification number in the reconciliation report.

To use the RREN primary command, enter:

```
RREN n;
```

in the command area.

where *n* is an integer identifying the proposed member's identification number in the reconciliation report.

The effects of the two forms of the RREN command are the same but you can only enter line commands when working in an interactive environment. You can enter several RREN line commands at the same time.

A dialog buffer in which you specify the new name of the proposed member is displayed in response to each RREN command.

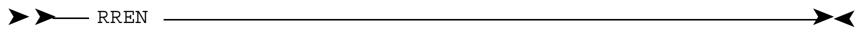
The reconciliation report is displayed by the RECONCILE command. To display the changes you have made with the RREN command enter a further RECONCILE command. You can also rename a proposed member by including a RENAMING keyword in a RECONCILE command.

Note:

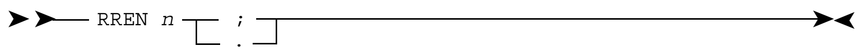
This command cannot be performed on a referred object.

Refer to the RECONCILE command for full details of renaming proposed members.

Syntax (Line Command)



Syntax (Primary Command)



where *n* is a proposed member's ID number in a reconciliation report.

RREP

RREP specifies that a proposed member documenting an external object replaces an existing repository member.

To use the RREP line command, enter:

```
RREP
```

in the line command area alongside the proposed member's identification number in the reconciliation report.

To use the RREP primary command, enter:

```
RREP n;
```

in the command area.

where *n* is an integer identifying the proposed member's identification number in the reconciliation report.

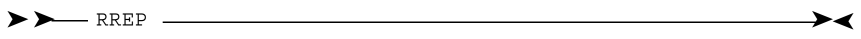
The effects of the two forms of the RREP command are the same but you can only enter line commands when working in an interactive environment. You can enter several RREP line commands at the same time.

The reconciliation report is displayed by the RECONCILE command. To display the changes you have made with the RREP command enter a further RECONCILE command.

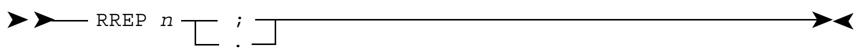
You can also specify that you want a proposed member to replace an existing repository member by including a REPLACING keyword in the RECONCILE command.

Refer to the RECONCILE command for full details of replacing existing members with proposed members.

Syntax (Line Command)



Syntax (Primary Command)



where *n* is a proposed member's ID number in a reconciliation report.

RUPD

RUPD updates an existing repository member from a reconciliation report in order to interactively change its source record.

To use the RUPD line command, enter:

```
RUPD
```

in the line command area alongside the identification number in the reconciliation report of the proposed member with the same name as the existing member.

To use the RUPD primary command, enter:

```
RUPD n;
```

in the command area.

where *n* is the identification number in the reconciliation report of a proposed member with the same name as an existing member.

The reconciliation report is displayed by the RECONCILE command. To display the changes you have made with the RUPD command enter a further RECONCILE command including the INITIALIZE keyword.

The effects of the two forms of the RUPD command are the same. The RUPD command opens a buffer in Update mode containing a copy of the source record of the selected repository member which you can then update interactively. You can only enter RUPD commands when working in an interactive environment.

If the selected member is an ITEM, you can copy the form-description and USAGE attribute of the proposed member into the update buffer. To do this, use the I, F, and P line commands in a command interface environment, or the X, A, and B line commands in a panel interface environment.

To enter the contents of the buffer into the repository use the FILE or SFILE commands. To abandon the update without adding the contents to the repository use the QUIT or XQUIT commands.

You can enter several RUPD line commands at the same time. The different update buffers are stacked. Use the QUERY ACTIVE-BUFFERS command to find out which buffers you have opened. The number of update buffers you can stack is determined by the buffer limit set in the repository by the Systems Administrator. Use the QUERY BUFFER-LIMIT command to find out the buffer limit.

Note:
The line commands only copy the form-description and USAGE attribute of the proposed member corresponding to the existing member at the top of the buffer stack.

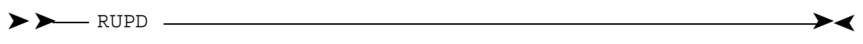
Having filed or quit the update buffer, you will go to an update buffer lower in the buffer stack or return to the reconciliation report.

The current status must be an update status. If the member does not exist in the current status, then the RUPD command copies the source record of the member from the next visible status in which it does exist. If you subsequently FILE or SFILE the member it is entered in the current status.

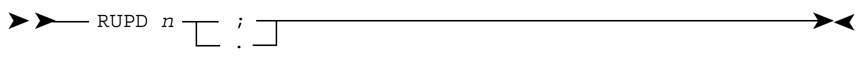
Refer to the *ASG-ControlManager User' Guide* for details of the FILE, SFILE, QUIT, XQUIT, I, F, P, and QUERY commands.

Refer to the *ASG-MethodManager Workstation User's Guide* for details of the X, A, and B line commands.

Syntax (Line Command)



Syntax (Primary Command)



where *n* is a proposed member's ID number in a reconciliation report.

TRANSLATE Command

This is the full syntax for the DictionaryManager TRANSLATE command.

```
TRANSLATE MANAGER-DICTIONARY
      ⎧ [KEPT [IN name]] [GENERIC] member-type
      ⎩ MEMBER member-name
       TO ⎧ IDD                 ⎫
          ⎨ IDMS                ⎬
          ⎩ OTHER-DICTIONARY    ⎭
       ONTO ⎧ PUBLIC  ⎫  mp-user [ ⎧REPLACE⎫ ]
            ⎩ PRIVATE ⎭            ⎩APPEND ⎭
            [USING mp-tr-rule] [PRINT] ⎧ ; ⎫
                                       ⎩ . ⎭
```

where:

name is the name of the KEPT-DATA list of members selected for translation.

member-type is a member type interrogation keyword valid on the current dictionary. This may be any valid basic member type or any UDS member type based on the basic member types.

member-name is the name of the dictionary member to be translated.

mp-user is the name of an MP-AID to be used for a USER-MEMBER. The name must be no more than 10 characters long. A public USER-MEMBER can be accessed by someone with a different logon ID; a private member cannot.

mp-tr-rule is the name of a TR-RULE member on the MP-AID. An *mp-tr-rule* specified here will override any default established by a previously issued SET command. The SET command is related to the target dictionary and member type to which you are translating, and is defined separately for each target dictionary system.

If the MEMBER clause is specified, the USING clause is required.

The PRINT option causes the USER-MEMBER holding the translated source to print automatically.

Refer to "Using the TRANSLATE Command" on page 34 for details of the TRANSLATE command.

TRANSFER Command

This is the full syntax of the DictionaryManager TRANSFER command.

```
TRANSFER
  FROM USER-MEMBER mp-user [LOGON logon-id]
  TO FILE filename ⎧ PARTITIONED                     ⎫
                   ⎩ SEQUENTIAL ['control-card']⎭
  [AS library-name]  ⎧ ; ⎫
                     ⎩ ; ⎭
```

where:

mp-user is the name of a USER-MEMBER on the MP-AID that holds the source input statements for the target dictionary.

filename is the name of the output source library dataset. It is the logical file name (ddname or dtfname) used in job control statements to indicate the external dataset name (physical file name) of the file to which the USER-MEMBER is to be copied.

control-card is a character string of up to 72 characters that is a library system control card image. Trailing spaces are implied. A single question mark (?) may be used to indicate the point at which the generated library member name is to be inserted in the control card.

library-name is the name to be given to the generated library member in the output dataset. The name must not be more than 16 characters long. The first character must be alphabetic, #, £, %, @, or a local currency symbol with the internal hex code 5B.

The order of the FROM USER-MEMBER and TO FILE clauses can be reversed.

PARTITIONED applies to the OS environment and defines the dataset to be a BPAM partitioned dataset.

SEQUENTIAL defines the dataset to be a QSAM sequential dataset

The AS clause declares a name under which the copied source statements are to be catalogued in the output source library dataset. If you omit this, *library-name* defaults to the name of the MP-AID USER-MEMBER being transferred.

Appendix A
Importing a COBOL Program

Introduction

This appendix is an example of importing from an external source. The standard four stages of the import procedure (extract, reconcile, preview, and populate) are shown. The example starts by extracting a COBOL program (held as a CMS file), and finishes by adding members representing that program to the repository.

The example external file and the example import executive routines described below are available in your Manager Products demonstration repository (status MPL). By constructing the executive routines onto the MP-AID and moving the example external file into a real external file you can actually perform the following example.

Refer to Chapter 1, "Introduction to DictionaryManager," on page 1 for details of the import procedure.

Refer to "Executive Routine Listings" on page 108 for listings of the example import executive routines.

Refer to "The Example External File" on page 122 for listing of the external file itself.

The Extract Stage

First define the ddname of the file to extract:

```
CMS FILEDEF COBIN DISK SDD11 COBOL A;
```

Next enter the EXTRACT command, specifying the input source (an external file), the ddname of the file, and the extract executive routine:

```
EXTRACT EXTERNAL-FILE COBIN USING UEXT001;
```

Output from the above command would be:

```
+-----------------------------------------------------------------+
   DM05702I   EXTRACTED 1 PARENT OBJECT(S) FROM EXTERNAL FILE
   DM05703I   EXTRACTED PROGRAM SDD11
------------------------------------------------------------------+
```

The example extract executive routine consists of nested routines to extract information from the PROCEDURE DIVISION of a COBOL program.

Figure 8 on page 104 illustrates the structure of the executive routine.

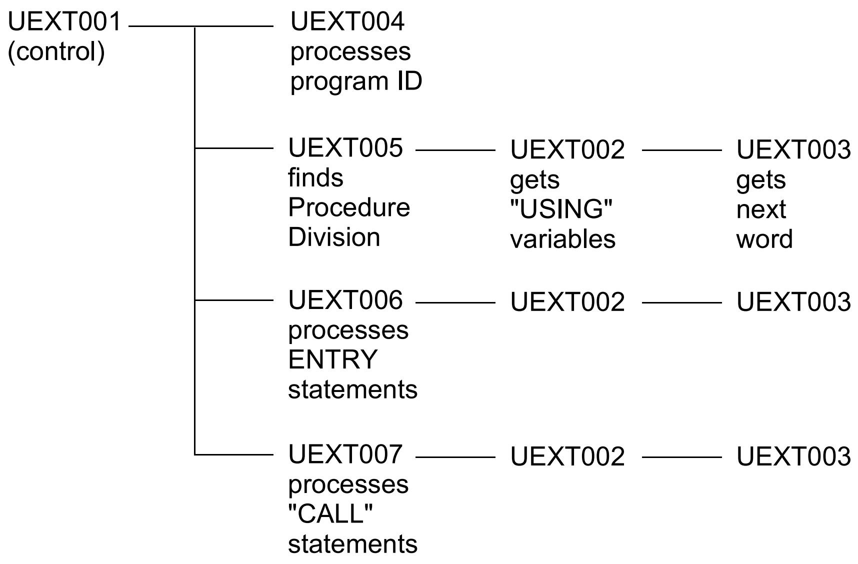


Figure 8. Structure of the Example Extract Executive Routine

The Reconcile Stage

After extracting data, enter the RECONCILE command, specifying the reconcile executive routine UREC001 to be used:

```
RECONCILE INITIALIZE USING UREC001;
```

Refer to "UREC001" on page 119 for a listing of the reconcile executive routine UREC001.

Output from the above command would be:

```
+----------------------------------------------------------------+
  *************************************************************
                Reconciliation summary report
           for extract of SDD11 from EXTERNAL FILE.
  *************************************************************
  ID  Proposed Member name          Type          Upd Condition
  -------------------------------------------------------------
  1   PG-SDD11                      PROGRAM       ADD
  2   GR-XMR-DATA                   GROUP         *   *    DUM
  3   GR-DMR-INTERFACE              GROUP         *   *    DUM
  4   GR-DOUTPUT                    GROUP         *   *    DUM
  5   GR-DICTIONARY-INPUT-LINES     GROUP         *   *    DUM
  8   GR-PCBNAME                    GROUP         *   *    DUM
  9   MO-SMU12                      MODULE        *   *    DUM
  11  GR-MESSAGE-DETAILS            GROUP         *   *    DUM
  12  GR-PERFORM                    GROUP         *   *    DUM
  29  MO-MPSCI                      MODULE        *   *    DUM
+----------------------------------------------------------------+
```

The Preview Stage

Next enter the PREVIEW command specifying the preview executive routine UPRE001 to be used:

```
PREVIEW IMPORT USING UPRE001;
```

Refer to "UPRE001" on page 120 for a listing of the preview executive routine UPRE001.

Output from the above command would be:

```
; -------------------------------------------------------
PG-SDD11
 ADD PG-SDD11;
 PROGRAM
 ALIAS
   COBOL  "SDD11"
 NOTE
   "This member was extracted by SYSDB2 at 10.17.03 on 30 MAR 1990"
   "This member was extracted by SYSDB2 at 11.18.30 on 16 MAR 1990"
 ENTRY "   "
 PARAMETERS GR-XMR-DATA
           ,GR-DMR-INTERFACE
           ,GR-DOUTPUT
           ,GR-DICTIONARY-INPUT-LINES
 ENTRY "ABCDEFGH"
 PARAMETERS GR-DMR-INTERFACE
           ,GR-DOUTPUT
 ENTRY "PQRSTUVW"
 PARAMETERS GR-PCBNAME
 CALLS MO-SMU12
 PASSING GR-XMR-DATA
        ,GR-MESSAGE-DETAILS
        ,GR-PERFORM
 CALLS MO-SMU12
 PASSING GR-XMR-DATA
        ,GR-MESSAGE-DETAILS
        ,GR-PERFORM
 CALLS MO-SMU12
 PASSING GR-XMR-DATA
        ,GR-MESSAGE-DETAILS
        ,GR-PERFORM
 CALLS MO-SMU12
 PASSING GR-XMR-DATA
        ,GR-MESSAGE-DETAILS
        ,GR-PERFORM
 CALLS MO-SMU12
 PASSING GR-XMR-DATA
        ,GR-MESSAGE-DETAILS
        ,GR-PERFORM
 CALLS MO-MPSCI
 PASSING GR-DMR-INTERFACE
        ,GR-DOUTPUT
        ,GR-DICTIONARY-INPUT-LINES
 ;
```

The Populate Stage

Finally, add the proposed definitions to the dictionary, by entering this command:

```
POPULATE FROM BUFFER;
```

Output from the above command would be:

```
DM00700I   DICTIONARY ... SUCCESSFULLY DEFINED
 DM01221I   PG-SDD11 EXISTS IN STATUS ...... AS A PROGRAM
 DM01132I   PG-SDD11 SUCCESSFULLY REPLACED
 DM01296I   ENCODING OF PG-SDD11
    00100     PROGRAM
    00200     ALIAS
    00300       COBOL   "SDD11"
    00400     NOTE
    00500     "This member was extracted by SYSDB2 at 10.17.03 on
30 MAR 1990"
    00600     "This member was extracted by SYSDB2 at 11.18.30 on
16 MAR 1990"
    00700     ENTRY "   "
    00800     PARAMETERS GR-XMR-DATA
    00900               ,GR-DMR-INTERFACE
    01000               ,GR-DOUTPUT
    01100               ,GR-DICTIONARY-INPUT-LINES
    01200     ENTRY "ABCDEFGH"
    01300     PARAMETERS GR-DMR-INTERFACE
    01400               ,GR-DOUTPUT
    01500     ENTRY "PQRSTUVW"
    01600     PARAMETERS GR-PCBNAME
    01700     CALLS MO-SMU12
    01800     PASSING GR-XMR-DATA
    01900            ,GR-MESSAGE-DETAILS
    02000            ,GR-PERFORM
    02100     CALLS MO-SMU12
    02200     PASSING GR-XMR-DATA
    02300            ,GR-MESSAGE-DETAILS
    02400            ,GR-PERFORM
    02500     CALLS MO-SMU12
    02600     PASSING GR-XMR-DATA
    02700            ,GR-MESSAGE-DETAILS
    02800            ,GR-PERFORM
    02900     CALLS MO-SMU12
    03000     PASSING GR-XMR-DATA
    03100            ,GR-MESSAGE-DETAILS
    03200            ,GR-PERFORM
    03300     CALLS MO-SMU12
    03400     PASSING GR-XMR-DATA
    03500            ,GR-MESSAGE-DETAILS
    03600            ,GR-PERFORM
    03700     CALLS MO-MPSCI
    03800     PASSING GR-DMR-INTERFACE
    03900            ,GR-DOUTPUT
    04000            ,GR-DICTIONARY-INPUT-LINES
 DM01280I   PG-SDD11 SUCCESSFULLY ENCODED
 DM00701I   DICTIONARY ... SUCCESSFULLY TERMINATED
```

Executive Routine Listings

UEXT001

```
mpxx
literal :
/*---------------------------------------------------------------
/* UEXT001 - Control module for sample COBOL Extract
/* Check input exists:
/*---------------------------------------------------------------
    if ARRAYHI(:ext_record:) <= 0 then do
      say :NO INPUT TO BE PROCESSED:
      exit 8
    end
/*---------------------------------------------------------------
/* Set up globals:
/* Environment variables:
/*--------------------------------------
    drop ext_obj_occ
    drop ext_object_id
/*--------------------------------------
/* Generic variables for all objects:
/*--------------------------------------
    drop ext_obj_name
    drop ext_obj_id
    drop ext_obj_type
    drop ext_obj_parent_pointer
    drop ext_obj_chain
    drop ext_obj_chain_end
    drop dmr_mem_gen
/*--------------------------------------
/* PROGRAM object:
/*--------------------------------------
    drop ext_entry_occ              /* no of ENTRY attributes
    drop ext_calls_occ              /* no of CALLS attributes
    drop ext_entry_ptr              /* ptr to 1st ENTRY attribute
    drop ext_calls_ptr              /* ptr to 1st CALLS attribute
/*----------------------------------------
/* MODULE object:
/*----------------------------------------
    drop ext_using_occ              /* no of USING attributes
    drop ext_using_ptr              /* ptr to 1st USING attribute
/*----------------------------------------
/* PARAMETERS-ENTRY relationship:
/*----------------------------------------
    drop ext_entry_string           /* ENTRY "string"
    drop ext_parm_occ               /* no of PARAMETER members for this
                                    /* ENTRY point
    drop ext_parm_ptr               /* ptr to 1st PARAMETER member
/*----------------------------------------
/* CALLS-USING relationship:
/*----------------------------------------
    drop ext_cusg_occ               /* no of MODULE members for this
    drop ext_cusg_ptr               /* ptr to 1st module member
/*----------------------------------------
/* Now declare them as globals:
/*----------------------------------------
    global ext_obj_occ
    global ext_object_id
    global ext_obj_name
    global ext_obj_id
    global ext_obj_type
    global ext_obj_parent_pointer
    global ext_obj_chain
    global ext_obj_chain_end
    global dmr_mem_gen
    global ext_entry_occ
    global ext_entry_ptr
    global ext_calls_occ
    global ext_calls_ptr
    global ext_entry_string
    global ext_parm_occ
    global ext_parm_ptr
    global ext_using_occ
    global ext_using_ptr
    global ext_cusg_occ
    global ext_cusg_ptr
/*----------------------------------------------------------------
/* Start processing:
/*----------------------------------------------------------------
    command x         /* pointer to object being processed
    command y         /* pointer to relationship being processed
    ext_obj_occ = 0
    x = 0
    y = 0
    SETRES ;            /* set up reserved word list
/*---------------------------------------------------------------
/* Find the program id:
/*---------------------------------------------------------------
    UEXT004 ;
    if &ccod > 4 then exit 8
/*---------------------------------------------------------------
/* Find the PROCEDURE DIVISION:
/*---------------------------------------------------------------
    UEXT005 ;
    if &ccod > 4 then exit 8
/*---------------------------------------------------------------
/* Find the ENTRY statements:
/*---------------------------------------------------------------
    UEXT006 ;
    if &ccod > 4 then exit 8
/*---------------------------------------------------------------
/* Find the CALL statements:
/*---------------------------------------------------------------
    UEXT007 ;
    if &ccod > 4 then exit 8
  exit 0
```

UEXT002

```
mpxx
literal :
/*---------------------------------------------------------------
/* UEXT002 :
/* input  - none
/* output - command variable array USING containing all using words
/*---------------------------------------------------------------
    drop    using
    command using
    local   l u
    u = 0
   -uloop
    UEXT003 ;
    if found_word ne :: then do
      u         = u+1
     using(u) = STRIP(found_word,:T:,:.:)  /* strip trailing period
      if POS(:.:,found_word) > 0 then goto end_using
     if SEARCH(:RESERVE:,found_word,,,:M:) > 0 then goto end_using
      goto uloop
    end
   -end_using
    exit
```

UEXT003

```
mpxx
literal :
/*--------------------------------------------------------------
/* UEXT003 :
/* input   - via command variables:
/*          curr_line = current line number
/*          curr_word = current word
/*          eof_flag  = end of file flag: 0 = not eof
/*                                        1 = eof
/* output - command variable found_word = next word
/*--------------------------------------------------------------
    command curr_line curr_word
    command found_word
    if eof_flag = 1 then exit 4
   -next_word
    line = SUBSTR(ext_record(curr_line),1,72)  /* only cols 1 - 72
    found_word = WORD(LINE,curr_word+1)
    if found_word = :: then do
      curr_line = curr_line+1
      curr_word = 0
      if curr_line > ARRAYHI(:ext_record:) then goto end_of_file
      goto next_word
    end
    curr_word = curr_word+1
    exit 0          /* found_word ne ::
   -end_of_file
    eof_flag = 1
    exit 4          /* end of file reached
```

UEXT004

```
mpxx
literal :
/*---------------------------------------------------------------
/* UEXT004 :
/* Find the program id:
/* Look for string with PROGRAM-ID in col 8 on non-comment line:
/*---------------------------------------------------------------
    command x           /* pointer to object being processed
    command p           /* pointer to current parent object
    command y           /* pointer to relationship being processed
    local   j           /* pointer to line being processed
    local   str         /* string constant
    j = 0
    str = :PROGRAM-ID:
   -prog_id_search
    j = SEARCH(:ext_record:,str,j+1,,:p:)
    if j > 0 then do
      len                 = LENGTH(: :°°str)
      if SUBSTR(ext_record(j),7,len) = : :°°str then goto
prog_id_found
      goto prog_id_search
    end
    say str :NOT FOUND:
    exit 8
/*---------------------------------------------------
/* set up *one* PROGRAM object at (x):
/*---------------------------------------------------
   -prog_id_found
    x                     = x+1
    prog_name             = WORD(ext_record(j),2)
    ext_obj_occ           = ext_obj_occ+1
   ext_obj_name(x)     = STRIP(prog_name,:T:,:.:) /* strip trailing
dot
    ext_object_id         = ext_obj_name(x)
    ext_obj_id(x)         = ext_obj_name(x)
    ext_obj_type(x)       = :PROGRAM:
    dmr_mem_gen(x)        = :GEN:
    ext_obj_chain_end(x)  = x         /* start (x) chain
    ext_entry_occ(x)      = 0         /* no of ENTRY attributes
    ext_calls_occ(x)      = 0         /* no of CALLS attributes
    p                     = x         /* set current parent
    exit 0
```

UEXT005

```
mpxx
literal :
/*--------------------------------------------------------------
/* UEXT005:
/* Find PROCEDURE DIVISION:
/* Look for string with PROCEDURE DIVISION on non-comment line:
/*--------------------------------------------------------------
    command y p
    local x j str
    j   = 0
    str = :PROCEDURE DIVISION:
   -proc_div_search
    j = SEARCH(:ext_record:,str,j+1,,:p:)
    if j > 0 then do
      len                = LENGTH(: :°°str)
      if SUBSTR(ext_record(j),7,len) = : :°°str  then goto
proc_div_found
      goto proc_div_search
    end
    say str :NOT FOUND:
    exit 8
   -proc_div_found
    if WORD(ext_record(j),3) = :USING: then do
/*--------------------------------------------------------------
/* set up command variables for control of UEXT002,UEXT003:
/*--------------------------------------------------------------
      command curr_line curr_word
      curr_line = j
      curr_word = 4
      UEXT002 ;
/*--------------------------------------------------------------
/* set up *one* PARAMETERS-ENTRY relationship at (y)
/*--------------------------------------------------------------
      itot = ARRAYHI(:using:)
      i    = 1
      y    = y+1                        /* next empty slot
      if ext_entry_ptr(p) = :: then -
        ext_entry_ptr(p) = y            /* initialize parent ptr
      ext_entry_occ(p)                    = ext_entry_occ(p)+1
      ext_entry_string(y)                = :   :  /* (y) attribute
      ext_parm_occ(y)                     = 0
     x              = ARRAYHI(:ext_obj_name:)+1  /* next object no
     ext_parm_ptr(y) = x                        /* initialize ptr
     -uloop
      if i <= itot then do
        ext_parm_occ(y)                     = ext_parm_occ(y)+1
        ext_obj_name(x)                     = using(i)
        ext_obj_type(x)                     = :GROUP:
        dmr_mem_gen(x)                      = :REF:
```

```
  ext_obj_parent_pointer(x)        = p /* chain(x) to parent(p)
    ext_obj_chain(ext_obj_chain_end(p)) = x
    ext_obj_chain_end(p)                = x
    x = x+1
    i = i+1
    goto uloop
  end
end
exit 0
```

UEXT006

```
mpxx
literal :
/*--------------------------------------------------------------
/* UEXT006: Find ENTRY statements:
/* Look for string with ENTRY in area B on non-comment line:
/*--------------------------------------------------------------
    command y p
    local x j str
    j   = 0
    str = :ENTRY :
   -entry_search
    j = SEARCH(:ext_record:,str,j+1,,:p:)
    if j > 0 then do
    if SUBSTR(ext_record(j),1,11) = :       : then goto entry_found
      goto entry_search
    end
/*--------------------------------------------------
/* This is the principal exit from UEXT006:
/*--------------------------------------------------
    exit 0
   -entry_found
    entry_string = WORD(ext_record(j),2)
/*--------------------------------------------------
/* set up *one* PARAMETERS-ENTRY relationship at (y)
/*--------------------------------------------------
      y    = y+1                          /* next empty slot
      if ext_entry_ptr(p) = :: then -
        ext_entry_ptr(p) = y              /* initialize parent ptr
      ext_entry_occ(p)                      = ext_entry_occ(p)+1
      ext_entry_string(y)                   = entry_string
      ext_parm_occ(y)                       = 0
    if WORD(ext_record(j),3) = :USING: then do
     x                = ARRAYHI(:ext_obj_name:)+1 /* next object no
     ext_parm_ptr(y)   = x                        /* initialize ptr

/*--------------------------------------------------------------
/* set up command variables for control of UEXT002,UEXT003:
/*--------------------------------------------------------------
      command curr_line curr_word
      curr_line = j
      curr_word = 3
      UEXT002 ;
/*--------------------------------------------------------------
/* set up USING objects:
/*--------------------------------------------------------------
      itot = ARRAYHI(:using:)
      i    = 1
      ext_parm_ptr(y)     = x        /* initialise ptr
     -uloop
```

```
  if i <= itot then do
    ext_parm_occ(y)                     = ext_parm_occ(y)+1
    ext_obj_name(x)                     = using(i)
    ext_obj_type(x)                     = :GROUP:
    dmr_mem_gen(x)                      = :REF:
  ext_obj_parent_pointer(x)        = p /* chain(x) to parent(p)
    ext_obj_chain(ext_obj_chain_end(p)) = x
    ext_obj_chain_end(p)                = x
    x = x+1
    i = i+1
    goto uloop
  end
end
goto entry_search                /* keep searching for ENTRY
```

UEXT007

```
mpxx
literal :
/*--------------------------------------------------------------
/* UEXT007:
/* Find CALL statements:
/* Look for string with CALL in area B on non-comment line:
/*--------------------------------------------------------------
    command y p
    local x j str
    j   = 0
    str = :CALL :
   -calls_search
    j = SEARCH(:ext_record:,str,j+1,,:p:)
    if j > 0 then do
      if WORD(ext_record(j),1) ne :CALL: then goto calls_search
    if SUBSTR(ext_record(j),1,11) = :        : then goto calls_found
      goto calls_search
    end
/*--------------------------------------------------------------
/* This is the principal exit from UEXT007:
/*--------------------------------------------------------------
    exit 0
   -calls_found
    calls_name = WORD(ext_record(j),2)
    q = POS(:":,calls_name)
    if q > 0 then do
      len        = LENGTH(calls_name)-2
      calls_name = SUBSTR(calls_name,2,len)
    end
/*--------------------------------------------------------------
/* set up *one* CALLs relationship at (y):
/*--------------------------------------------------------------
      y                    = y+1         /* next empty slot
      if ext_calls_ptr(p)  = :: then -
        ext_calls_ptr(p)  = y            /* initialize parent ptr
      ext_calls_occ(p)                     = ext_calls_occ(p)+1
      x                   = ARRAYHI(:ext_obj_name:)+1
      ext_cusg_occ(y)     = 1
      ext_cusg_ptr(y)     = x
      ext_obj_name(x)                      = calls_name
      ext_obj_type(x)                      = :MODULE:
      dmr_mem_gen(x)                       = :REF:
     ext_obj_parent_pointer(x)          = p /* chain(x) to parent(p)
      ext_obj_chain(ext_obj_chain_end(p)) = x
      ext_obj_chain_end(p)                 = x
      ext_using_occ(x)                     = 0
    if WORD(ext_record(j),3) = :USING: then do
      c                  = x+1              /* next object no
      ext_using_ptr(x)   = c                /* initialise ptr
```

```
/*--------------------------------------------------------------
/* set up command variables for control of UEXT002,UEXT003:
/*--------------------------------------------------------------
      command curr_line curr_word
      curr_line = j
      curr_word = 3
      UEXT002 ;
/*--------------------------------------------------------------
/* set up USING objects at (c):
/*--------------------------------------------------------------
      itot = ARRAYHI(:using:)
      i    = 1
     -uloop
      if i <= itot then do
        ext_using_occ(x)                = ext_using_occ(x)+1
        ext_obj_name(c)                 = using(i)
        ext_obj_type(c)                 = :GROUP:
        dmr_mem_gen(c)                  = :REF:
      ext_obj_parent_pointer(c)        = p /* chain(c) to parent(p)
        ext_obj_chain(ext_obj_chain_end(p)) = c
        ext_obj_chain_end(p)            = c
        c = c+1
        i = i+1
        goto uloop
      end
    end
    goto calls_search                /* keep searching for CALLS
```

UREC001

```
mpxx
/* ---------------------------------------------------------- *
/* UREC001 : Supplied reconcile exec                          *
/*                                                            *
/* input parameter : &p0 = current object number.             *
/*                                                            *
/* This exec will process external object types as follows:   *
/*                                                            *
/*  ext_obj_type    dmr_mem_name  dmr_mem_type (dmr_mem_gen)  *
/*  -------------------------------------------------------   *
/*  PROGRAM         PG-obj_name   PROGRAM      (GEN)          *
/*  MODULE          MO-obj_name   MODULE       (REF)          *
/*  GROUP           GR-obj_name   GROUP        (REF)          *
/*                                                            *
/* ---------------------------------------------------------- *
     local x
     literal :
     set x &p0
     if TYPE(dmr_mem_gen(x)) = :U: then exit
/* ---------------------------------------------PROGRAM---------
     if ext_obj_type(x) eq :PROGRAM: then do
        set dmr_mem_type(x) :PROGRAM:
        set dmr_mem_name(x) :PG-:ext_obj_name(x)
        exit
     end
/* ----------------------------------------------MODULE----------
     if ext_obj_type(x) eq :MODULE: then do
        set dmr_mem_type(x) :MODULE:
        set dmr_mem_name(x) :MO-:ext_obj_name(x)
        exit
     end
/*----------------------------------------------GROUP-----------
     if ext_obj_type(x) eq :GROUP: then do
        set dmr_mem_type(x) :GROUP:
        set dmr_mem_name(x) :GR-:ext_obj_name(x)
        exit
     end
/*----------------------------------------------ERROR-----------
/* No member-naming or typing rule found:
/*   issue warning message 5743 and take default action:
/*--------------------------------------------------------------
     MESSAGE 5743 W '' ext_obj_type(x) ext_obj_name(x)
     set dmr_mem_name(x) ext_obj_name(x)
     set dmr_mem_type(x) ext_obj_type(x)
     exit
```

UPRE001

```
mpxx
 literal :
/*---------------------------------------------------------------
/* UPRE001 - User-supplied layout control exec and layout
/*
/* input parameter  : x     = array number of member whose
/*                            definition is to be generated
/* output parameter : &ccod = 0 - member accepted and generated
/*                            4 - member rejected
/*                            8 - x not valid
/*---------------------------------------------------------------
     command x
     if TYPE(x) ne :N: then exit 8
     if dmr_mem_type(x) eq :PROGRAM: then do   /* do if module
       goto layout
      -control_exit
       exit 0
     end
     exit 4                                   /* reject otherwise
/*---------------------------------------------------------------
/*  Layout exec:
/*---------------------------------------------------------------
 -layout
     local i j itot jtot
     WRITEF dmr_mem_func(x) dmr_mem_name(x) ;
     WRITEF dmr_mem_type(x)
     UPRE002 ;     /* common clauses
/*--------------------------------------------------------------
/*  output ENTRY clauses:
/*--------------------------------------------------------------
     jtot = ext_entry_occ(x)
     if jtot > 0 then do
       y = ext_entry_ptr(x)
       j = 1
      -entryloop
       if j <= jtot then do
         entry_string = ext_entry_string(y)
         q = POS(:":,entry_string)
         if q = 0 then entry_string = :":°°entry_string°°:":
         WRITEF :ENTRY: entry_string
         itot = ext_parm_occ(y)
         if itot > 0 then do
           c = ext_parm_ptr(y)
           i = 1
           f = :PARAMETERS :
          -parmloop
           if i <= itot then do
             WRITEF f°°dmr_mem_name(c)
             f = :          ,:
```

```
             i = i+1
             c = c+1
             goto parmloop
           end
           j = j+1
           y = y+1
           goto entryloop
         end
       end
     end
/*--------------------------------------------------------------
/*  output CALLS clauses:
/*--------------------------------------------------------------
     jtot = ext_calls_occ(x)
     if jtot > 0 then do
       y = ext_calls_ptr(x)
       j = 1
      -callsloop
       if j <= jtot then do
         z = ext_cusg_ptr(y)
         WRITEF :CALLS: dmr_mem_name(z)
         itot = ext_using_occ(z)
         if itot > 0 then do
           c = ext_using_ptr(z)
           i = 1
           f = :PASSING :
          -passingloop
           if i <= itot then do
             WRITEF f°°dmr_mem_name(c)
             f = :       ,:
             i = i+1
             c = c+1
             goto passingloop
           end
           j = j+1
           y = y+1
           goto callsloop
         end
       end
     end
/*--------------------------------------------------------------
/*  Terminator:
/*--------------------------------------------------------------
     WRITEF :;:
     goto control_exit
```

The Example External File

This is a listing of the COBOL program to be imported.

```
IDENTIFICATION DIVISION.                                        SDD00010
 -------------------------                                      SDD00020
     SKIP2                                                      SDD00030
 PROGRAM-ID.                  SDD11.                            SDD00040
     SKIP1                                                      SDD00050
 AUTHOR.                      JOHN DENT.                        SDD00060
     SKIP1                                                      SDD00070
 DATE-WRITTEN.                SEPT 1982.                        SDD00080
     SKIP1                                                      SDD00090
 DATE-COMPILED.               00/00/00.                         SDD00100
     SKIP1                                                      SDD00110
 INSTALLATION.                MSP.                              SDD00120
     SKIP1                                                      SDD00130
 REMARKS.         DESCRIPTION                                   SDD00140
               "DATAMANAGER INTERFACE - GET PCBS".              SDD00150
             THIS MODULE FORMATS AND PASSES TO DATAMANAGER      SDD00160
             THE COMMANDS NEEDED TO BUILD A LIST OF PCBS.       SDD00170
              IF NO PCBS ARE FOUND, DI-RETURN ID SET TO         SDD00180
            THE VALUE WITH CONDITION-NAME DICT-CALL-FAILED.SDD00190
 ENVIRONMENT DIVISION.                                          SDD00200
 ----------------------                                         SDD00210
     SKIP2                                                      SDD00220
 CONFIGURATION SECTION.                                         SDD00230
 -----------------------                                        SDD00240
     SKIP1                                                      SDD00250
 SOURCE-COMPUTER.             IBM-370.                          SDD00260
     SKIP1                                                      SDD00270
 OBJECT-COMPUTER.             IBM-370.                          SDD00280
     EJECT                                                      SDD00290
 DATA DIVISION.                                                 SDD00300
 ---------------                                                SDD00310
     SKIP1                                                      SDD00320
 WORKING-STORAGE SECTION.                                       SDD00330
 ------------------------                                       SDD00340
     SKIP1                                                      SDD00350
 /INCLUDE XMRMDET C170                                          SDD00360
    05 MD-VARIABLE            PIC X(5).                         SDD00370
 01  PCB-LIST-WORK.                                             SDD00380
    05 PCB-COUNT-WORK         PIC S9(4)      COMP.              SDD00390
    05 PCB-ENTRY-WORK         OCCURS 100 TIMES.                 SDD00400
      10 PCB-NAME-WORK        PIC X(32).                        SDD00410
      10 PCB-TYPE-WORK        PIC X(5).                         SDD00420
 01  PCB-SUB                  PIC S9(4)      COMP.              SDD00430
 01  PCB-SUB-WORK             PIC S9(4)      COMP.              SDD00440
 01  PCB-ENQUIRY-NAME         PIC X(32).                        SDD00450
     EJECT                                                      SDD00460
 LINKAGE SECTION.                                               SDD00470
 ----------------                                                SDD00480
     SKIP1                                                       SDD00490
 /INCLUDE XMRDATA C170                                           SDD00500
     EJECT                                                       SDD00510
 /INCLUDE XMRDMRI C170                                           SDD00520
     EJECT                                                       SDD00530
 /INCLUDE XMRDOUT C170                                           SDD00540
     EJECT                                                       SDD00550
 /INCLUDE COBDINP C170                                           SDD00560
     EJECT                                                       SDD00570
  PROCEDURE DIVISION USING                                       SDD00580
                  XMR-DATA                                       SDD00590
                  DMR-INTERFACE                                  SDD00600
                  DOUTPUT                                        SDD00610
                 DICTIONARY-INPUT-LINES.                         SDD00620
 ----------------------------------------                        SDD00630
     SKIP2                                                       SDD00640
  SDD11-GET-PCBS.                                                SDD00650

****************************************************************
*SDD00660
 *   INITIALIZATION.                                            *SDD00670

****************************************************************
*SDD00680
     MOVE SPACES TO PCB-LIST.                                    SDD00690
     MOVE 0 TO PCB-COUNT.                                        SDD00700
     MOVE SPACES TO PCB-LIST-WORK.                               SDD00710
     MOVE 0 TO PCB-COUNT-WORK.                                   SDD00720
     MOVE SPACES TO DICT-LINE (1).                               SDD00730
     SKIP1                                                       SDD00740

****************************************************************
*SDD00750
 *    SET DMR-INPUT-AREA-LENGTH SO THAT ONE LINE WILL BE READ.
*SDD00760
 *   PERFORM SUBROUTINE TO GET PCB'S.                           *SDD00770

****************************************************************
*SDD00780
    ENTRY "ABCDEFGH" USING DMR-INTERFACE                         SDD00790
                        DOUTPUT.                                 SDD00800
     MOVE DMR-INPUT-LINE-LENGTH TO DMR-INPUT-AREA-LENGTH.
SDD00810
    PERFORM GET-PCBS THRU GET-PCBS-EXIT.                         SDD00820
     SKIP1                                                       SDD00830
  SDD11-GOBACK.                                                  SDD00840
     GOBACK.                                                     SDD00850
     EJECT                                                       SDD00860
  GET-PCBS.                                                      SDD00870
```

```
*****************************************************************
*SDD00880
 *    SET UP COMMAND TO FIND ALL PCB MEMBERS THAT ARE USED
*SDD00890
 *    DIRECTLY BY THE PROGRAM AND PERFORM SUBROUTINE TO PLACE
*SDD00900
 *    THEM IN THE PCB LIST. IF NO PCBS, SET DICT-CALL-FAILED.
*SDD00910

*****************************************************************
*SDD00920
      MOVE "WHICH" TO DLP1-WHICH (1).                          SDD00930
      MOVE CP-PCB-NAME TO DLP1-PCB-MT (1).                     SDD00940
      MOVE "DIRECTLY" TO DLP1-DIR (1).                         SDD00950
      MOVE "CONSTITUTE" TO DLP1-CONST (1).                     SDD00960
      MOVE RTP-PROGRAM-NAME TO DLP1-NAME (1).                  SDD00970
      MOVE ";" TO DLP1-TERM (1).                               SDD00980
      PERFORM BUILD-PCB-LIST THRU BUILD-PCB-LIST-EXIT.
SDD00990
     IF PCB-COUNT - 0                                          SDD01000
        MOVE "F" TO DI-RETURN.                                 SDD01010
     SKIP1                                                     SDD01020
  GET-PCBS-EXIT.                                               SDD01030
     EXIT.                                                     SDD01040
     EJECT                                                     SDD01050
  BUILD-PCB-LIST.                                              SDD01060
     ENTRY "PQRSTUVW" USING PCBNAME.                           SDD01070

*****************************************************************
*SDD01080
 *   CALL DICTIONARY TO PASS COMMAND.                         *SDD01090

*****************************************************************
*SDD01100
     MOVE "1" TO DMR-FUNCTION.                                 SDD01110
      PERFORM CALL-DICTIONARY THRU CALL-DICTIONARY-EXIT.
SDD01120

*****************************************************************
*SDD01130
 *   CALL DICTIONARY TO RETURN INFORMATION.                   *SDD01140

*****************************************************************
*SDD01150
     MOVE "2" TO DMR-FUNCTION.                                 SDD01160
  READ-PCB-LIST.                                               SDD01170
      PERFORM CALL-DICTIONARY THRU CALL-DICTIONARY-EXIT.
SDD01180
     IF NOTHING-RETURNED                                       SDD01190
     OR DMR-ENDED                                              SDD01200
        GO TO COMBINE-LISTS.                                   SDD01210
     IF MESSAGE-RETURNED                                     SDD01220
         GO TO READ-PCB-LIST.                                SDD01230

*****************************************************************
*SDD01240
 *    ADD OUTPUT MESSAGE PCBS DIRECTLY TO PCB-LIST. OUTPUT
*SDD01250
 *    MESSAGE IF PCB-LIST OVERFLOWS, OR AN INACCESSIBLE MEMBER IS
*SDD01260
 *   ENCOUNTERED.                                            *SDD01270

*****************************************************************
*SDD01280
     IF NOT OUTPUT-MESSAGE-PCB                               SDD01290
         GO TO BUILD-WORK-LIST.                              SDD01300
     ADD 1 TO PCB-COUNT.                                     SDD01310
     IF PCB-COUNT > 100                                      SDD01320
         MOVE 30031 TO MD-NUMBER                             SDD01330
         MOVE "E" TO MD-SEVERITY                             SDD01340
         MOVE " " TO MD-INDICATOR                            SDD01350
         MOVE 0 TO MD-LENGTH                                 SDD01360
       CALL "SMU12" USING XMR-DATA MESSAGE-DETAILS           SDD01370
       PERFORM ERROR-BOUNCE THRU ERROR-BOUNCE-EXIT           SDD01380
         MOVE "F" TO DI-RETURN                               SDD01390
         GO TO BUILD-PCB-LIST-EXIT.                          SDD01400
     IF DPQBDMEM - SPACES                                    SDD01410
        IF DPQBTMEM - "INACCESSIBLE MEMBER"                  SDD01420
             MOVE 30048 TO MD-NUMBER                         SDD01430
             MOVE DPQBTOT TO MD-VARIABLE                     SDD01440
             MOVE 5 TO MD-LENGTH                             SDD01450
             MOVE "E" TO MD-SEVERITY                         SDD01460
             MOVE " " TO MD-INDICATOR                        SDD01470
           CALL "SMU12" USING XMR-DATA MESSAGE-DETAILS       SDD01480
           PERFORM ERROR-BOUNCE THRU ERROR-BOUNCE-EXIT       SDD01490
             SUBTRACT 1 FROM PCB-COUNT                       SDD01500
             GO TO READ-PCB-LIST.                            SDD01510
    MOVE DPQBDMEM TO PCB-NAME (PCB-COUNT).                   SDD01520
    MOVE DPQBDMTP TO PCB-TYPE (PCB-COUNT).                   SDD01530
     GO TO READ-PCB-LIST.                                    SDD01540
 BUILD-WORK-LIST.                                            SDD01550

*****************************************************************
*SDD01560
 *    ADD PCBS OTHER THAN OUTPUT MESSAGE PCBS TO WORK LIST FOR
*SDD01570
 *    LATER ADDITION TO PCB-LIST. OUTPUT MESSAGE IF WORK LIST
*SDD01580
 *    OVERFLOWS OR AN INACCESSIBLE MEMBER IS ENCOUNTERED.
*SDD01590
 *    FOR STRUCTURE PCB'S, REPLACE TYPE FIELD WITH THE KEYLENGTH
*SDD01600
```

```
 *   OF THE PCB IF PRESENT, OTHERWISE BY SPACES.           *SDD01610

*****************************************************************
*SDD01620
     IF NOT STRUCTURE-PCB                                   SDD01630
     AND NOT GSAM-PCB                                       SDD01640
        GO TO READ-PCB-LIST.                                SDD01650
     ADD 1 TO PCB-COUNT-WORK.                               SDD01660
     IF PCB-COUNT-WORK > 100                                SDD01670
        MOVE 30031 TO MD-NUMBER                             SDD01680
        MOVE "E" TO MD-SEVERITY                             SDD01690
        MOVE " " TO MD-INDICATOR                            SDD01700
        MOVE 0 TO MD-LENGTH                                 SDD01710
       CALL "SMU12" USING XMR-DATA MESSAGE-DETAILS          SDD01720
       PERFORM ERROR-BOUNCE THRU ERROR-BOUNCE-EXIT          SDD01730
        MOVE "F" TO DI-RETURN                               SDD01740
        GO TO BUILD-PCB-LIST-EXIT.                          SDD01750
     IF DPQBDMEM - SPACES                                   SDD01760
        IF DPQBTMEM - "INACCESSIBLE MEMBER"                 SDD01770
           MOVE 30048 TO MD-NUMBER                          SDD01780
           MOVE DPQBTOT TO MD-VARIABLE                      SDD01790
           MOVE 5 TO MD-LENGTH                              SDD01800
           MOVE "E" TO MD-SEVERITY                          SDD01810
           MOVE " " TO MD-INDICATOR                         SDD01820
          CALL "SMU12" USING XMR-DATA MESSAGE-DETAILS       SDD01830
          PERFORM ERROR-BOUNCE THRU ERROR-BOUNCE-EXIT       SDD01840
           SUBTRACT 1 FROM PCB-COUNT-WORK                   SDD01850
           GO TO READ-PCB-LIST.                             SDD01860
      MOVE DPQBDMEM TO PCB-NAME-WORK (PCB-COUNT-WORK).
SDD01870
      MOVE DPQBDMTP TO PCB-TYPE-WORK (PCB-COUNT-WORK).
SDD01880
     GO TO READ-PCB-LIST.                                   SDD01890
  COMBINE-LISTS.                                            SDD01900

*****************************************************************
*SDD01910
 *    MOVE STRUCTURE AND GSAM PCBS TO PCB LIST FOLLOWING OUTPUT
*SDD01920
 *    MESSAGE PCBS. THUS ALL PCBS ARE RETAINED IN THEIR ORIGINAL
*SDD01930
 *    ORDER ACCORDING TO TYPE, BUT OUTPUT MESSAGE PCBS COME
FIRST.*SDD01940
 *   OUTPUT MESSAGE IF PCB-LIST OVERFLOWS.                 *SDD01950

*****************************************************************
*SDD01960
     IF PCB-COUNT-WORK - 0                                  SDD01970
        GO TO BUILD-PCB-LIST-EXIT.                          SDD01980
    IF ((PCB-COUNT + PCB-COUNT-WORK) > 100)                 SDD01990
        MOVE 30031 TO MD-NUMBER                             SDD02000
```

```
          MOVE "E" TO MD-SEVERITY                           SDD02010
          MOVE " " TO MD-INDICATOR                          SDD02020
          MOVE 0 TO MD-LENGTH                               SDD02030
        CALL "SMU12" USING XMR-DATA MESSAGE-DETAILS         SDD02040
        PERFORM ERROR-BOUNCE THRU ERROR-BOUNCE-EXIT         SDD02050
          MOVE "F" TO DI-RETURN                             SDD02060
          GO TO BUILD-PCB-LIST-EXIT.                        SDD02070
      MOVE 0 TO PCB-SUB-WORK.                               SDD02080
  COMBINE-LOOP.                                             SDD02090
      ADD 1 TO PCB-SUB-WORK.                                SDD02100
      IF PCB-SUB-WORK > PCB-COUNT-WORK                      SDD02110
          GO TO BUILD-PCB-LIST-EXIT.                        SDD02120
      ADD 1 TO PCB-COUNT.                                   SDD02130
      MOVE PCB-NAME-WORK (PCB-SUB-WORK) TO PCB-NAME (PCB-COUNT).
SDD02140
      MOVE PCB-TYPE-WORK (PCB-SUB-WORK) TO PCB-TYPE (PCB-COUNT).
SDD02150

*****************************************************************
*SDD02160
 *    FOR STRUCTURE PCBS, REPLACE TYPE WITH THE KEYLENGTH IF ANY,
*SDD02170
 *   OTHERWISE WITH SPACES.                                *SDD02180

*****************************************************************
*SDD02190
      IF PCB-TYPE (PCB-COUNT) - "STRUC"                     SDD02200
        PERFORM GET-KEYLENGTH THRU GET-KEYLENGTH-EXIT.      SDD02210
      GO TO COMBINE-LOOP.                                   SDD02220
      SKIP1                                                 SDD02230
  BUILD-PCB-LIST-EXIT.                                      SDD02240
      EXIT.                                                 SDD02250
      EJECT                                                 SDD02260
  GET-KEYLENGTH.                                            SDD02270

*****************************************************************
*SDD02280
 *    OBJECT - MOVE KEYLENGTH IF PRESENT TO TYPE-FIELD, OTHERWISE
*SDD02290
 *   SET TYPE FIELD TO SPACES.                             *SDD02300

*****************************************************************
*SDD02310
     MOVE SPACES TO PCB-TYPE (PCB-COUNT).                   SDD02320

*****************************************************************
*SDD02330
 *   PASS A "REPORT PCB-NAME" COMMAND TO DMR.              *SDD02340

*****************************************************************
*SDD02350
```

```
     MOVE PCB-NAME (PCB-COUNT) TO PCB-ENQUIRY-NAME.         SDD02360
  REPORT-PCB.                                               SDD02370
      MOVE SPACES TO DICT-LINE (1).                         SDD02380
      MOVE "REPORT" TO DLR-REPORT (1).                      SDD02390
      MOVE PCB-ENQUIRY-NAME TO DLR-NAME (1).                SDD02400
      MOVE ";" TO DLR-TERM (1).                             SDD02410
      MOVE "1" TO DMR-FUNCTION.                             SDD02420
       PERFORM CALL-DICTIONARY THRU CALL-DICTIONARY-EXIT.
SDD02430

*****************************************************************
*SDD02440
 *    RETURN INFORMATION.                                  *SDD02450

*****************************************************************
*SDD02460
      MOVE "2" TO DMR-FUNCTION.                             SDD02470
       PERFORM CALL-DICTIONARY THRU CALL-DICTIONARY-EXIT.
SDD02480

*****************************************************************
*SDD02490
 *     IF A "DEFINITION AS IN" CLAUSE IS ENCOUNTERED, THE
*SDD02500
 *     DEFINITION OF THE PCB IS IN ANOTHER PCB. GET NAME OF OTHER
*SDD02510
 *     PCB AND LOOP BACK TO REPEAT REPORT AND SEARCH FOR KEYLENGTH.
*SDD02520

*****************************************************************
*SDD02530
  KEYLENGTH-LOOP.                                           SDD02540
       PERFORM CALL-DICTIONARY THRU CALL-DICTIONARY-EXIT.
SDD02550
      IF DPDLADEF - "DEFINITION AS IN"                      SDD02560
         MOVE DPDLANAM TO PCB-ENQUIRY-NAME                  SDD02570
          GO TO REPORT-PCB.                                 SDD02580
      IF NOTHING-RETURNED                                   SDD02590
      OR DMR-ENDED                                          SDD02600
          GO TO GET-KEYLENGTH-EXIT.                         SDD02610

*****************************************************************
*SDD02620
 *   MOVE KEYLENGTH VALUE IF FOUND TO PCB-TYPE.            *SDD02630

*****************************************************************
*SDD02640
      IF DPDLSKLK - "KEYLENGTH"                             SDD02650
         MOVE DPDLSKLN2 TO PCB-TYPE (PCB-COUNT)             SDD02660
      ELSE                                                  SDD02670
          GO TO KEYLENGTH-LOOP.                             SDD02680
```

```
    SKIP1                                                       SDD02690
 GET-KEYLENGTH-EXIT.                                            SDD02700
    EXIT.                                                       SDD02710
    EJECT                                                       SDD02720
 CALL-DICTIONARY.                                               SDD02730

*****************************************************************
*SDD02740
 *   CALL DMRUS TO COMMUNICATE WITH DMR.                       *SDD02750

*****************************************************************
*SDD02760
     CALL "MPSCI" USING DMR-INTERFACE                           SDD02770
                        DOUTPUT                                 SDD02780
                      DICTIONARY-INPUT-LINES.                   SDD02790
    SKIP1                                                       SDD02800
    PERFORM EROR-REPORTED THRU EROR-REPORTED-EXIT.              SDD02810
 CALL-DICTIONARY-EXIT.                                          SDD02820
    EXIT.                                                       SDD02830
    EJECT                                                       SDD02840
 EROR-REPORTED.                                                 SDD02850
 *                                                              SDD02860

*****************************************************************
*SDD02870
 *    IF DMR HAS REPORTED A SERIOUS ERROR, SET ERROR CODE TO
*SDD02880
 *   TRIGGER OFF ERROR-BOUNCING                                *SDD02890

*****************************************************************
*SDD02900
    IF  FATAL-ERROR                                             SDD02910
      MOVE "EROR" TO PS-RETURN-2                                SDD02920
      GO TO SDD11-GOBACK.                                       SDD02930
 EROR-REPORTED-EXIT.                                            SDD02940
    EXIT.                                                       SDD02950
    EJECT                                                       SDD02960
 ERROR-BOUNCE.                                                  SDD02970
 *                                                              SDD02980

*****************************************************************
*SDD02990
 *    IF MPSCI HAS REPORTED LINE LIMIT EXCEEDED OR NO CORE
*SDD03000
 *   OR A SERIOUS DMR ERROR                                    *SDD03010
 *   DURING THIS COMMAND RETURN TO CALLING MODULE              *SDD03020
 *    (PS-RETURN-2 - 'LINE'/'CORE'/'EROR' - PS-RETURN-2-ERR )
*SDD03030

*****************************************************************
*SDD03040
```

```
           IF  PS-RETURN-2-ERR                                   SDD03050
             GO TO SDD11-GOBACK.                                 SDD03060
       ERROR-BOUNCE-EXIT.                                        SDD03070
           EXIT.                                                 SDD03080
           EJECT                                                 SDD03090
```

Appendix B
Importing an Entity-relational Model

Introduction

This is an example designed to illustrate how to import an entity-relational model from version 3.10 of the BACHMAN/Analyst tool into a Manager Products repository.

The example follows the four standard stages in the import procedure: extract, reconcile, preview, and populate. Information in an export file generated by BACHMAN/Analyst is used as the basis for the import procedure.

This example applies to users who have the panel interface. To keep the example reasonably simple, not all modeling aspects are reproduced into the Manager Products repository.

All executive routines associated with this example are available in your Manager Products demonstration dictionary. The import file itself is supplied as a separate dataset. Refer to your installation manual for details of this dataset.

The example illustrates several aspects of generic import:

- Extracting data both from an external file and from a variable
- Writing extract, reconcile, and preview Executive Routines
- Some features of the Manager Products Procedures Language and some useful techniques you can exploit

Refer to "Listings of Executive Routines" on page 144 for listings of the example import executive routines.

Refer to "The Imported Model" on page 157 for a diagram of the imported model.

The Import File

The import file used in this example was generated using the export facility of the Bachman Analyst Workbench. The file is formatted according to rules specific to the Bachman Analyst Workbench. It contains a great deal of information, but only a small portion of it is of interest to this example. While its format is complex and very detailed, its general characteristics are not, and these are described below.

The file represents a complete model and is made up of sections, each of which describes a particular aspect of the model. For example, there is a section describing the entities in the model. These sections are described below.

Each section has an identifier string (enclosed in double asterisks) at the start, followed by one or more lines of data. Each line has fields separated by commas. Each field contains a value that represents either a number, a text-string, or a time stamp. The numeric values you will be extracting are all integers. The text strings are delimited by double quotes. Time stamps have a more complex format that will not described, as they will not be extracted in this example.

Figure 9 on page 132 shows how entities, partnerships, and partnership sets are related. Entities are shown as boxes with rounded corners, partnerships (relationships) are shown as diamonds, and partnership sets (links) are shown as lines linking a diamond with a box.

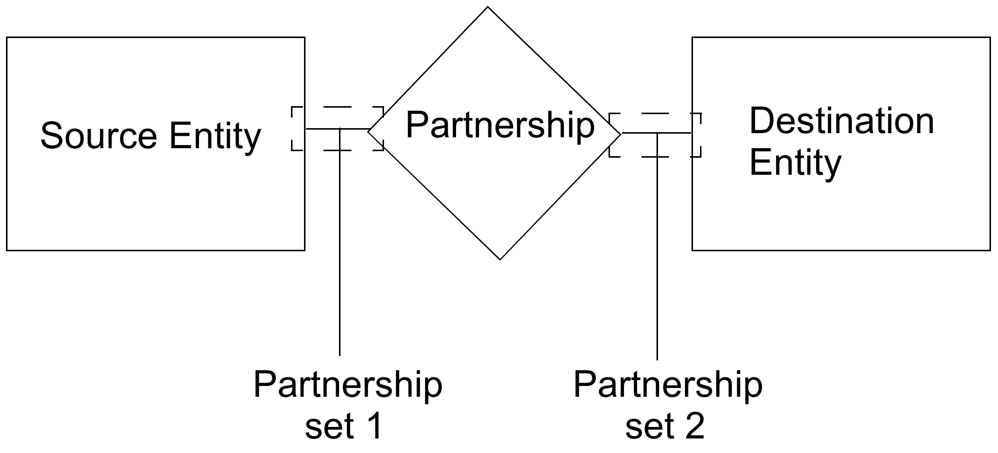


Figure 9. Illustration of Relationships

In order to derive relationships, information about the entities, the partnerships, and the partnership sets must be combined. Each line of the file has other common fields including:

- The key, a number that identifies the object uniquely in the file (and therefore the model)
- The name of the creator of the object and a corresponding time stamp (not imported)
- The name of the person who last modified the object and a corresponding time stamp (not imported)

The names and explanation for each of the extracted fields below is taken from the Bachman documentation.

The Enterprise Model

Identifying string: `**ENTERPRISE_MODEL**`

Fields Extracted	
enterprise_modelke	Unique number that identifies the object
name	Name of object

The Entities

Identifying string: `**ENTITY**`

Fields Extracted	
entitykey	Unique number that identifies the object
name	Name of object
minvol	Lowest possible number of occurrences for this entity
maxvol	Highest possible number of occurrences for this entity
expvol	Expected number of occurrences for this entity
grwpct	Expected growth percentage for this entity
gptu	Character representing the growth rate period for this entity
normlvl	Character representing the normalisation level
normok	Character indicating that the entity is sufficiently nomalized

The Partnerships

Identifying string: `**PARTNERSHIP**`

Fields Extracted	
partnershipkey	Unique number that identifies this object
partnership_set	Partnership_setkey value that identifies the source partnership set of the partnership
partnership_set2	Partnership_setkey value that identifies the destination partnership set of the partnership
min1vol	Minimum volume of the source partnership set for this partnership
max1vol	Maximum volume of the source partnership set for this partnership
min2vol	Minimum volume of the destination partnership set for this partnership
max2vol	Maximum volume of the destination partnership set for this partnership

The Partnerships Sets

Identifying string: `**PARTNERSHIP_SET**`

Fields Extracted	
partnership_setkey	Unique number that identifies this object
name	Name of object
minvol	Minimum volume for this partnership set
maxvol	Maximum volume for this partnership set
expvol	Expected volume for this partnership set

Mapping the Two Models

The model we imported from the Bachman Analyst Workbench is derived according to the rules or schema for that tool. The Manager Products repository has a different schema. It is therefore important to establish how the Bachman model maps onto the Manager Products Repository Information Model (RIM).

The RIM is determined by the meta-schema used for it. In this case, there are these correspondences:

Bachman Analyst Workbench	Manager Products Repository
Enterprise Model	GROUP
Entity	ENTITY
Partnership + Partnership Set	BUSINESS-RELATIONSHIP

Notice that the GROUP member type models the whole enterprise model. This member is just a convenient way to group together all of the ENTITY and BUSINESS-RELATIONSHIP members using a SEE clause to facilitate future repository interrogation.

The Extract Stage

Introduction

The import file described in the previous section must be imported into the Manager Products environment.

You can often extract data by issuing an EXTRACT command directly and specifying your own extract Executive Routine in the USING keyword. This is done in the example in Appendix A, "Importing a COBOL Program," on page 103, where data from a COBOL source program is imported.

In this example, however, several, more complex EXTRACT commands are needed. The EXTRACT commands are therefore grouped together in a user executive routine (with other additional processing) which is executed instead.

Processing Overview

The extract stage is initiated by calling the top-level executive routine, by entering this command:

```
BACHMAN;
```

This routine calls other user executive routines as shown in Figure 10 on page 135:

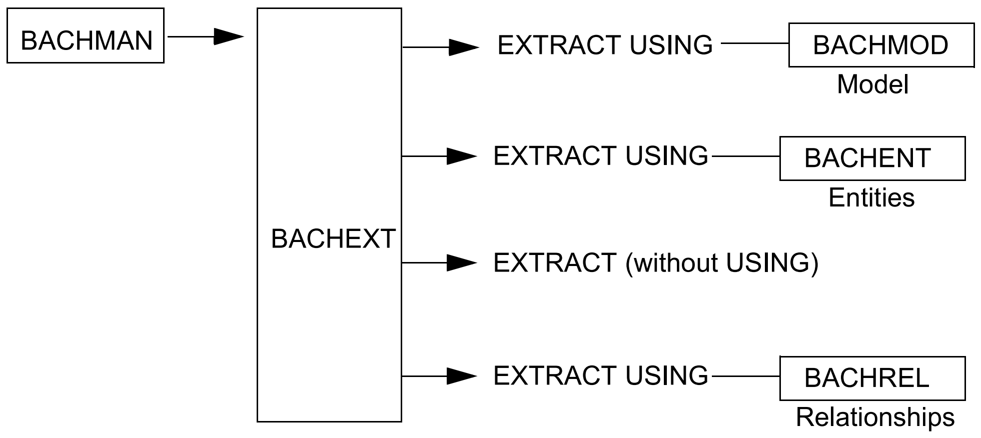


Figure 10. Extracting from the Bachman Analysis Workbench

The EXTRACT command is performed four times, once for each object type imported. Because the commands are very similar, they are all performed by the executive routine BACHEXT which receives as input details for performing the command.

BACHMOD, BACHENT, and BACHREL are executive routines invoked via the USING keyword in the relevant EXTRACT command. Each executive called this way is then responsible for collecting the data extracted from the import file and processing it for the subsequent reconcile stage.

BACHMOD assembles the model, BACHENT assembles the entities, and BACHREL assembles the relationships. BACHREL assembles data from two EXTRACT commands, as it must combine partnership and partnership-set data.

The data must be structured in such a way that it can be recognized by the subsequent reconcile stage.

The executive routines used in the extract stage of import from the example are described in the following sections.

Extract Executive Routines

Coding Style

Throughout the example, these coding standards apply:

- Comments heading main sections are placed between dashed lines.
- Blocks of processing are indented by two characters for each successive level of nesting.
- Alignment and white space in groups of similar lines within logical blocks is designed to highlight similarity of coding between the lines.
- Wherever possible typing is structured to ease the addition of new but similar lines.
- Calls to Manager Products commands and other executives are prefixed by the directive MPR.
- Literal delimiters (the colon character) are used to prevent inadvertent variable substitution.

Data Structuring using VECTORS

Groups of variable arrays often need to be considered together as part of one larger structure. For example, extracted object names are stored in the variable array EXT_OBJ_NAME while the corresponding types are stored in the array EXT_OBJ_TYPE.

The first element of EXT_OBJ_NAME (the name of the first extracted object), is related to the first element of EXT_OBJ_TYPE (the type of the first extracted object). In general, the `i`th object has its attributes stored in the `i`th elements of several arrays.

It is useful to group these related arrays (EXT_OBJ_NAME, EXT_OBJ_TYPE, and others, in this example), so that they can be treated as parts of one larger aggregate.

This grouping is achieved by defining a new variable array called a vector. The elements of a vector contain the names of the variable arrays which are to be grouped together. The order of the grouping, if it is significant, is reflected in the ordering of the names stored in the vector.

Thus, suppose you are storing three attributes for each extracted object; the name in EXT_OBJ_NAME, the type in EXT_OBJ_TYPE, and the generate option (whether to generate a proposed member for import) in DMR_MEM_GEN.

To treat the three variable arrays as an aggregate, set up a vector (called BACHMAN_OBJ_VECTOR here, but it can have any name), containing:

- EXT_OBJ_NAME as element 1
- EXT_OBJ_TYPE as element 2
- DMR_MEM_GEN as element 3

This is shown in Figure 11 on page 137.

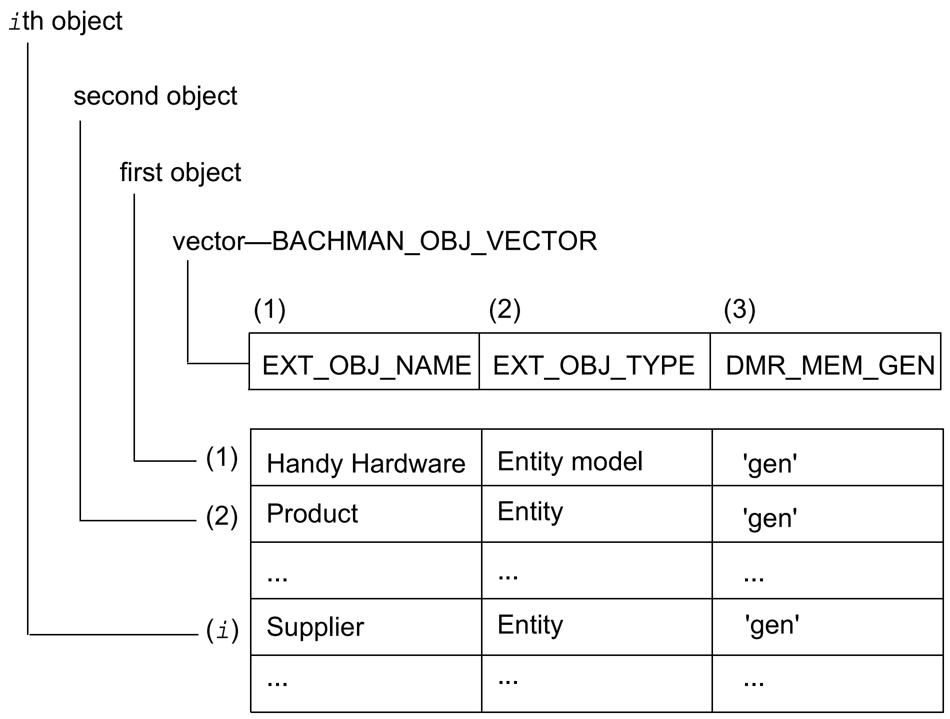


Figure 11. Vector Example

BACHMAN_OBJ_VECTOR is now treated as a single aggregate.

First, you can display it as an aggregate in a tabular manner by using the COMMAND member MPDYDISVCT, to show the inter-relationships of the constituent arrays. Each row of the display corresponds to all the values of a particular extracted object—the `i`th row reflects the `i`th extracted object. This is easier to analyze than output from the VLIST directive.

Second, you can set up processing loops more easily and ensure that uniform treatment is applied to all variables in the vector. For example, you declare all variables as command variables by setting up a loop to do the declarations (thus removing a common cause of errors). Also, if you need to add variables to the vector, you need do it in only one place. ASG uses the INTERPRET directive to execute the DROP and COMMAND directives on variables whose names are not needed.

Third, you can exploit other COMMAND members which operate on vectors. The BACHMAN executive uses MPDYVCTDIR to declare as global all command variables in the vector BACHMAN_OBJ_VECTOR.

The objects represented by the vector can and do have different attributes represented by different variables contained in the vector. Thus, an entity will have a minvol attribute while a relationship will have a primary and inverse verb. That presents no problem; for the entity, merely leave as null the array elements which do not form part of it.

This is illustrated in Figure 12 on page 138.

EXT_OBJ_TYPE	MINIVOL	VERB1	VERB2
Entity	30	null	null
Relationship	null	sells	sold by
Entity	100	null	null
etc...			

Figure 12

BACHMAN

This is the top-level executive routine that controls the entire extract stage.

It declares the vector BACHMAN_OBJ_VECTOR that is used in several parts of the extract stage. It then sets up parameters and calls the executive BACHEXT to extract from the external file. Four calls are made to BACHEXT, one for each main section of the import file.

When all extracting is complete, the extract vector BACHMAN_OBJ_VECTOR and its contained variable arrays are redeclared as global (without altering contents of variables) to make these arrays available to the reconcile stage.

The messages DM05702I and DM05703I are then issued to indicate what was extracted.

Finally, there is a section which will display the BACHMAN_OBJ_VECTOR for debugging purposes. It needs to be enabled as indicated in the code to be activated. You are encouraged to do so in order to see what a vector display looks like.

BACHEXT

This routine issues four EXTRACT commands, differing according to the input parameters start, genspec, and using. After the extract it returns control to the routine that called it (BACHMAN).

The ddname for the external file is fixed as DD1, so you must have a job-control card with ddname DD1 which identifies the Bachman Analyst export file.

There is debugging code in this executive which you are encouraged to activate as indicated in the source code. When you do this, you will get a vector display of the data extracted during the current invocation of the routine. (This is not the same as the consolidated extract data defined in BACHMAN_OBJ_VECTOR).

Notice how the vector MPGEN_VECTOR is set up. Since the input parameter genspec contains the names of the variables to be extracted with the EXTRACT command, this information is used to set up MPGEN_VECTOR.

First we break down genspec into its constituent words (the names of the extract variables) using PARSE VALUE genspec WITH genword():

so if genspec contains:

```
'e_entitykey . . . . e_name e_minvol etc'
```

genword will contain (in its elements):

```
e_entitykey
.
.
.
.
e_name
e_minvol
etc.
```

(Remember, the periods were used to denote fieldsnot to be extracted by the EXTRACT command).

Second, remove the periods via another EXTRACT command, specifying a SKIPR parameter to ignore array elements which contain periods. The result is placed in MPGEN_VECTOR which will now contain the names of the command variables used in the EXTRACT command issued previously, with no intervening periods.

The vector MPGEN_VECTOR can now be displayed using the COMMAND member MPDYDISVCT.

BACHMOD, BACHENT, BACHREL

These executives are called through the USING clause of the EXTRACT command.

Each is responsible for transferring data extracted by the EXTRACT command and placing it in the consolidated object structure represented by the vector BACHMAN_OBJ_VECTOR. Additional chaining data must also be set up for the subsequent reconcile stage.

BACHMOD and BACHENT are relatively simple as each extracted object maps directly into one consolidated object.

BACHREL is more complex, because it must combine data from two EXTRACT commands. One set contains partnership data and the other set contains partnership-set data. These are combined into relationships in what constitutes the core of the mapping or transformation from the Bachman to the Manager Products models.

The Reconcile Stage

Introduction

When the Extract stage is complete, the RECONCILE command accesses the extracted data and performs additional processing to propose member names and member types.

The user-written reconcile executive BACHREC is used for this. The executive is called repeatedly, once for each extracted object which is to be proposed as a member for the repository.

Starting the Reconciliation

The reconciliation stage is started by issuing the following RECONCILE command:

```
RECONCILE INITIALISE USING BACHREC;
```

BACHREC, the executive routine invoked, is relatively simple. It performs some basic name-editing to conform to Manager Products repository rules. Notice the call to the name-reduction function REDUCE. To ensure that the proposed member name will not exceed the permitted 32-character limit after the two-character member-type prefix is added, limit the name to 29 characters. The SUBSTR function is then used to ignore the first three characters returned by REDUCE (which contain a return code value) and the STRIP command is used to strip off trailing blanks.

The executive then examines the type of object for which it has been invoked. Remember, it is invoked repeatedly for every object extracted.

Based on the object type (which can be a model, an entity, or a relationship), it proposes an appropriate member name and member type.

The strategy for proposing relationship names has been kept deliberately simple. In practice, reconciling relationships can get complex depending on the type of reconciliation desired. Using a simple name will not, in general, reveal the existence of an already existing (but differently named) relationship linking the same two entities in the repository.

The strategy adopted in this example, however, should ensure uniqueness within the scope of one enterprise model.

The Preview Stage

Introduction

For the purposes of this example, assume that output is to be displayed only on the screen, from which you can populate the repository directly. This is about the generation of the data definition statements from the data extracted and processed so far.

To start the preview stage, calling the preview executive routine BACHPREV, enter:

```
PREVIEW IMPORT USING BACHPREV;
```

Note:

If you attempt to import the model again, you will get different results during the reconcile stage, since it will find the members proposed for import already in the repository.

An overview of the modular structure of the Preview stage is shown in Figure 13 on page 141.

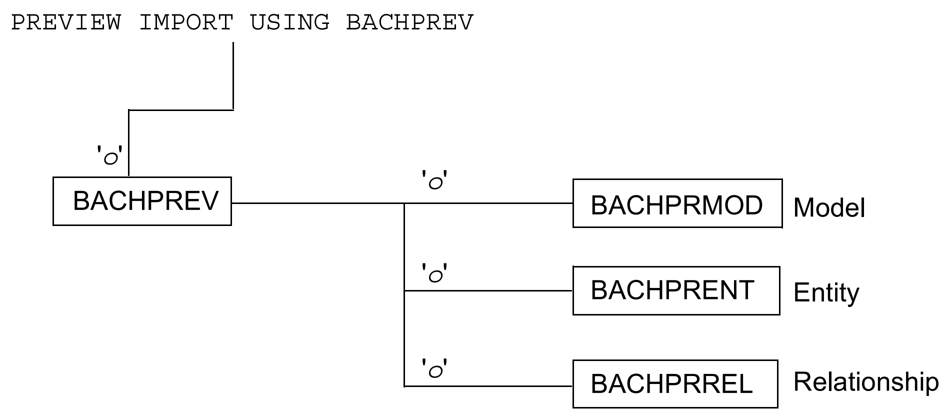


Figure 13

o is the object number passed as a parameter.

Preview Executive Routines

BACHPREV

This routine receives control when you issue the PREVIEW IMPORT command as shown.

The BACHPREV routine is responsible for identifying the type of the object represented by the parameter passed to it. It then branches to the appropriate preview executive as shown in the diagram, passing on the object number as a parameter.

BACHPREV is called once for each proposed object. This makes writing the routine easier; all loop-control functions are taken care of by generic import functions.

BACHPRMOD

This routine is responsible for generating the command- and member-definition statement for the model as a whole. Remember, the input parameter passed to it is the number of the object, and this is used as an index to the variable arrays resulting from the previous import stages.

The WRITEF directive is used to generate the output. You need not concern yourself with the destination of the output (that is, whether output has been directed to the screen, a user member, or some other external dataset). That is taken care of by the PREVIEW command and any optional ONTO clause you specify.

The repository update command, the proposed member name, and the member type are generated first. Note the variable names which contain these values. They will have been set up by the previous reconcile stage.

The corporate executive routine MPDYMMLOCC is called to generate the common clauses. Existing common clauses may have been extracted for members already in the repository, and these will be preserved. The parameters passed to MPDYMMLOCC are:

- The object number
- An alias type of SHORT-NAME to indicate that we wish to add a new alias of that type to any existing aliases (if the proposed member already existed in the repository)

Next, the SEE clause is generated by following the parent-child chain which was set up during the extract stage (by the extract executive BACHMOD).

Finally, the terminator—a single semicolon—is generated.

BACHPRENT

This generates the definition for the ENTITY member type. It is very simple, and uses the values of the entity attributes that have been extracted to generate the member clauses.

BACHPRREL

This generates the definition for the BUSINESS-RELATIONSHIP member type. It too is simple as most of the complex model mapping will have been completed during the extract stage.

The names of the source and target entities are derived by using the pointer variables EXT_OBJ_REL_SRC_ENT_PTR and EXT_OBJ_REL_TRG_ENT_PTR.

The pointers are used as index values into the array DMR_MEM_NAME which contains the repository member names for the entities.

The Populate Stage

Introduction

After completing preview, the repository is populated by entering:

```
POPULATE FROM BUFFER ROLLBACK;
```

The ROLLBACK option is not mandatory but is advisable. It causes the populate stage to be treated as a single logical unit of work (LUW). If any errors occur during this stage, the LUW is rolled back as one unit, leaving the repository unchanged. If errors do occur, it is much easier to examine the preview output and determine the cause of errors before a second attempt at populating, without having to remove partial updates to the repository.

Validating the Model

Finally, because an entity-relationship model is subject to additional constraints imposed by the repository schema, you should validate the model using the VALIDATE command.

First, collect together (in a KEPT-DATA list) all the entities and business-relationships which constitute the model by interrogating on the constituents of the model. (You can now see the usefulness of the GROUP member also imported):

```
KEEP WHICH MEMBERS CONSTITUTE EM-HANDY-HARDWARE-COMPANY VIA SEE
DIRECTLY;
```

You can confirm that the members in the list need validating by using the commands:

```
SET CHECK-CHAR ?;

LIST KEPT-DATA;
```

The first command sets the question mark as an indicator character that shows up on the member list (displayed by the second command). Each member entry will contain a question mark as the last character under the Condition heading.

Next, perform validation on the members in the list with:

```
PERFORM 'VALIDATE MEMBER "*"' KEPT-DATA;
```

This will validate all entities and business-relationships leaving you with a complete, validated model in the repository.

Listings of Executive Routines

BACHMAN

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHMAN - Top-level executive for EXTRACT phase of Bachman
/*            GENERIC IMPORT EXAMPLE
/*
/* NOTE - A valid JOB-CONTROL statement must be active to associate
/*         the external dataset with the ddname 'DD1'.
/*---------------------------------------------------------------
/*---------------------------------------------------------------
/* Define a command variable which refers to a group of related
/* command variables.  We call such a command variable a VECTOR.
/*
/* It can be thought of as a 'handle' to the related set of
/* command variables. It provides useful capabilities exploited
/* in this executive, such as:
/*    a) a quick way to make extracted command variables global.
/*    b) a powerful display capability (useful when debugging).
/*       Related sets of variables are grouped together
/*       in the display to show them as rows in a table.
/*       Such example code is included but disabled.
/*---------------------------------------------------------------
    drop    bachman_obj_vector                    /* empty first
    command bachman_obj_vector                   /* then redeclare
/*---------------------------------------------------------------
/* Common attributes for all objects:
/*---------------------------------------------------------------
bachman_obj_vector(1)   = :ext_obj_key:
    bachman_obj_vector(2)   = :ext_obj_name:
    bachman_obj_vector(3)   = :ext_obj_type:
    bachman_obj_vector(4)   = :dmr_mem_gen:
    bachman_obj_vector(5)   = :ext_obj_chain:
    bachman_obj_vector(6)   = :ext_obj_chain_end:
    bachman_obj_vector(7)   = :ext_obj_parent_pointer:
    bachman_obj_vector(8)   = :ext_obj_id:
    bachman_obj_vector(9)   = :dmr_mem_desc:
    bachman_obj_vector(10) = :ext_save_gen:
/*---------------------------------------------------------------
/* (No Enterprise model attributes),
/* Entity attributes:
/*---------------------------------------------------------------
    bachman_obj_vector(11) = :ext_obj_ent_minvol:
    bachman_obj_vector(12) = :ext_obj_ent_maxvol:
    bachman_obj_vector(13) = :ext_obj_ent_expvol:
    bachman_obj_vector(14) = :ext_obj_ent_grwpct:
    bachman_obj_vector(15) = :ext_obj_ent_gptu:
    bachman_obj_vector(16) = :ext_obj_ent_normlvl:
    bachman_obj_vector(17) = :ext_obj_ent_normok:
```

```
/*---------------------------------------------------------------
/* Relationship attributes:
/*---------------------------------------------------------------
    bachman_obj_vector(18) = :ext_obj_rel_src_ent_ptr:   /* source
    bachman_obj_vector(19) = :ext_obj_rel_src_min_card:
    bachman_obj_vector(20) = :ext_obj_rel_src_max_card:
    bachman_obj_vector(21) = :ext_obj_rel_src_med_card:
    bachman_obj_vector(22) = :ext_obj_rel_src_mandatory:
    bachman_obj_vector(23) = :ext_obj_rel_inv_verb:
    bachman_obj_vector(24) = :ext_obj_rel_trg_ent_ptr:   /* target
    bachman_obj_vector(25) = :ext_obj_rel_trg_min_card:
    bachman_obj_vector(26) = :ext_obj_rel_trg_max_card:
    bachman_obj_vector(27) = :ext_obj_rel_trg_med_card:
    bachman_obj_vector(28) = :ext_obj_rel_trg_mandatory:
    bachman_obj_vector(29) = :ext_obj_rel_fwd_verb:
/*---------------------------------------------------------------
/* Declare the 'vectored' variables as command:
/*---------------------------------------------------------------
do bachman_obj_vector()
      i = FDO(:DARRAY:)
      interpret :drop    :bachman_obj_vector(i)  /* 'empty' first
      interpret :command :bachman_obj_vector(i)  /*  then declare
    end
/*---------------------------------------------------------------
/* Extract parameters for enterprise model data:
/*---------------------------------------------------------------
    local start genspec using
    start   =        :**ENTERPRISE_MODEL**:       /* STARTA parm
    genspec =        : em_enterprise_modelke:     /* GENERATE parm
    genspec = genspec: .: /* createdby    - ignored
    genspec = genspec: .: /* createdtime  - ignored
    genspec = genspec: .: /* modifiedby   - ignored
    genspec = genspec: .: /* modifiedtime - ignored
    genspec = genspec: em_name:
    using   =        :BACHMOD:                    /* USING parm
    MPR :BACHEXT ':start:' ':genspec:' :using  ;
/*---------------------------------------------------------------
/* Extract entity data:
/*---------------------------------------------------------------
    start   =        :**ENTITY**:                 /* STARTA parm
    genspec =        : e_entitykey:               /* GENERATE parm
    genspec = genspec: .: /* createdby    - ignored
    genspec = genspec: .: /* createdtime  - ignored
    genspec = genspec: .: /* modifiedby   - ignored
    genspec = genspec: .: /* modifiedtime - ignored
    genspec = genspec: e_name:
    genspec = genspec: e_minvol:
    genspec = genspec: e_maxvol:
    genspec = genspec: e_expvol:
    genspec = genspec: e_grwpct:
    genspec = genspec: e_gptu:
    genspec = genspec: e_normlvl:
    genspec = genspec: e_normok:
                          /* rest          - ignored
    using   =        :BACHENT:                    /* USING parm
    MPR :BACHEXT ':start:' ':genspec:' :using  ;
/*----------------------------------------------------------------
/* Extract PARTNERSHIP data:
/*----------------------------------------------------------------
    start   =        :**PARTNERSHIP**:            /* STARTA parm
    genspec =        : p_partnershipkey:          /* GENERATE parm
    genspec = genspec: .: /* createdby    - ignored
    genspec = genspec: .: /* createdtime  - ignored
    genspec = genspec: .: /* modifiedby   - ignored
    genspec = genspec: .: /* modifiedtime - ignored
    genspec = genspec: p_partnership_set:
    genspec = genspec: p_partnership_set2:
    genspec = genspec: p_min1vol:
    genspec = genspec: p_max1vol:
    genspec = genspec: p_min2vol:
    genspec = genspec: p_max2vol:
    MPR :BACHEXT ':start:' ':genspec:' ;:   /* NOTE - no USING exec
/*----------------------------------------------------------------
/* Extract PARTNERSHIP_SET data:
/*----------------------------------------------------------------
    start   =        :**PARTNERSHIP_SET**:        /* STARTA parm
    genspec =        : ps_partnership_setkey:     /* GENERATE parm
    genspec = genspec: .: /* createdby    - ignored
    genspec = genspec: .: /* createdtime  - ignored
    genspec = genspec: .: /* modifiedby   - ignored
    genspec = genspec: .: /* modifiedtime - ignored
    genspec = genspec: ps_entity:
    genspec = genspec: ps_name:
    genspec = genspec: ps_minvol:
    genspec = genspec: ps_maxvol:
    genspec = genspec: ps_expvol:
                          /* rest          - ignored
    using   =        :BACHREL:                    /* USING parm
    MPR :BACHEXT ':start:' ':genspec:' :using  ;
/*----------------------------------------------------------------
/* Save dmr_mem_gen in ext_save_gen:
/* (This permits subsequent RECONCILE INIT commands to restore
/*  extracted data without having to re-Extract).
/*----------------------------------------------------------------
    ext_save_gen() = dmr_mem_gen
/*----------------------------------------------------------------
/* Make bachman_obj_vector and all variables in it, global:
/*----------------------------------------------------------------
                    global bachman_obj_vector
    MPR :MPDYVCTDIR global bachman_obj_vector ;:
/*----------------------------------------------------------------
/* Display informatory messages of objects Extracted:
```

```
/*-----------------------------------------------------------------
    local tot o                            /* object number
          tot = ARRAYHI(:ext_obj_name:)    /* total objects
    MESSAGE 5702 :I: '' tot :EXTERNAL-FILE:

   do ext_obj_name()                  /* for all objects extracted
      o = FDO(:DARRAY:)
      MESSAGE 5703 :I: '' ext_obj_type(o) ext_obj_name(o)
    end
/*-----------------------------------------------------------------
/* Exit:
/*-----------------------------------------------------------------
    exit00
   exit 0  /* comment out this line to get diagnostics that follow
/*-----------------------------------------------------------------
/* Debug -
/* Display the objects - each (extracted) object is represented by
/* one row in the table representing the vector:
/*
/* MPDYDISVCT is the 'display vector' facility. It takes the
/*           following parameters:
/*           - the name of the VECTOR to be displayed
/*           - the number of rows to display
/*           - the number of characters which form the common
/*             prefix to the vectored variables. Set this to zero
/*-----------------------------------------------------------------
    local rows
    rows = ARRAYHI(:ext_obj_type:)
    MPR :MPDYDISVCT bachman_obj_vector: rows 0;
```

BACHEXT

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHEXT - Extract from file for Bachman example:
/*---------------------------------------------------------------/
*---------------------------------------------------------------
/* Locals:
/*---------------------------------------------------------------
    local start genspec
/*---------------------------------------------------------------
/* Input parms:
/*---------------------------------------------------------------
    parse arg start genspec using

    start   = STRIP(start,:B:,:’:)
    genspec = STRIP(genspec,:B:,:’:)
    using   = UPPER(using)

/*  if using ne :: then using = :USING ’:UPPER(using):’:
/*---------------------------------------------------------------
/* Extract Entities:
/*---------------------------------------------------------------
                        MPR : EXTRACT    EXTERNAL DD1      :
                        MPR : SEPARATOR    ’,’            :
                        MPR : GEN          ’:genspec:’    :
                        MPR : STARTA POS 1 ’:start:’      :
                        MPR : ENDF   POS 1 ’**’           :
    if using ne :: then MPR : USING        ’:using:’      :
                        MPR :;:

    exit 0  /* comment out this line to get diagnostics that follow
/*---------------------------------------------------------------
/* Set up display vector mpgen_vector, containing names of variables
/* extracted but ommitting periods (.):
/*---------------------------------------------------------------
    drop    genword mpgen_vector
    command genword mpgen_vector
/*---------------------------------------------------------------
/* Set up genword with everything in genspec (including periods):
/*---------------------------------------------------------------
    PARSE VALUE genspec WITH genword()
/*---------------------------------------------------------------
/* Use EXTRACT to set up mpgen_vector without periods:
/*---------------------------------------------------------------
    MPR : EXTRACT VARIABLE GENWORD ALL:
    MPR : GENERATE ’mpgen_vector’:
    MPR : SKIPA    ’.’            :
    MPR :;:
/*---------------------------------------------------------------
/* Display results:
```

```
/*---------------------------------------------------------------
    local rows
    rows =  ARRAYHI(mpgen_vector)
    MPR :MPDYDISVCT mpgen_vector: rows 0;
```

BACHMOD

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHMOD - Build ENTERPRISE MODEL object from extracted Bachman
data:
/*---------------------------------------------------------------
/*---------------------------------------------------------------
/* Declare locals:
/*---------------------------------------------------------------
    local o                  /* object index
    local em                 /* enterprise model index
/*---------------------------------------------------------------
/* Populate the EXTRACT variables:
/*---------------------------------------------------------------
    o = ARRAYHI(:ext_obj_name:)                /* last object slot

    do em_enterprise_modelke()
      o  = o+1                            /* next object slot
      em = FDO(:DARRAY:)                  /* current ent model
/*---------------------------------------------------------------
/* Set extracted object attributes (o) from entity_model (em):
/*---------------------------------------------------------------
      ext_obj_key(o)  = em_enterprise_modelke(em) /* object key
      ext_obj_name(o) = em_name(em)               /* object name
      ext_obj_type(o) = :ENTITYMODEL:             /* object type
     dmr_mem_gen(o)   = :GEN:                   /* generate member
      ext_obj_id(o)   = em_name(em)               /* object name
/*---------------------------------------------------------------
/* (No chaining needed since enterprise model is at head of chain)
/*---------------------------------------------------------------

    end

/*---------------------------------------------------------------
/* Set ext_object_id as a global for RECONCILE header:
/*
/*                    Reconciliation summary report
/*           for extract of MODEL '<name>'    from EXTERNAL-FILE.
/*                          -------------
/*                      --->ext_object_id<---
/*---------------------------------------------------------------
    global ext_object_id
           ext_object_id = :MODEL ':em_name(1):':
    exit 0
```

BACHENT

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHENT - Build ENTITY objects from extracted Bachman entity:
/*---------------------------------------------------------------
/*---------------------------------------------------------------
/* Declare locals:
/*---------------------------------------------------------------
    local o                 /* object index
    local e                 /* entity index
/*---------------------------------------------------------------
/* Populate the EXTRACT variables:
/*---------------------------------------------------------------
    o = ARRAYHI(:ext_obj_name:)          /* last object slot
    do e_entitykey()
      o = o+1                            /* next object slot
      e = FDO(:DARRAY:)                  /* current entity
/*---------------------------------------------------------------
/* Set common object attributes (o) from entity (e):
/*---------------------------------------------------------------
      ext_obj_key(o)      = e_entitykey(e)   /* object key
      ext_obj_name(o)     = e_name(e)        /* object name
      ext_obj_type(o)     = :ENTITY:         /* object type
      dmr_mem_gen(o)      = :GEN:            /* generate member
      ext_obj_id(o)       = e_name(e)        /* object name
/*---------------------------------------------------------------
/* Adjust chaining of object (o) to enterprise model (1):
/*---------------------------------------------------------------
      ext_obj_chain(o-1)        = o      /* previous obj to current
      ext_obj_chain_end(1)       = o      /* chain end to current
      ext_obj_parent_pointer(o) = 1      /* current to parent
/*---------------------------------------------------------------
/* Extracted attributes:
/*---------------------------------------------------------------
      ext_obj_ent_minvol(o)     = e_minvol(e)
      ext_obj_ent_maxvol(o)     = e_maxvol(e)
      ext_obj_ent_expvol(o)     = e_expvol(e)
      ext_obj_ent_grwpct(o)     = e_grwpct(e)
      ext_obj_ent_gptu(o)       = e_gptu(e)
      ext_obj_ent_normlvl(o)    = e_normlvl(e)
      ext_obj_ent_normok(o)     = e_normok(e)
    end
    exit 0
```

BACHREL

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHPRREL - PREVIEW executive for Bachman RELATIONSHIP:
/*---------------------------------------------------------------
    local o         /* object number
    parse arg o     /* get object number
/*---------------------------------------------------------------
/* Repository update function, member name, member type:
/*---------------------------------------------------------------
    WRITEF dmr_mem_func(o) dmr_mem_name(o) ;
    WRITEF dmr_mem_type(o)
/*---------------------------------------------------------------
/* Common clauses (alias type is 'SHORT'):
/*---------------------------------------------------------------
    MPR :MPDYMMLOCC :o: SHORT ;:
/*---------------------------------------------------------------
/* Relationship clauses:
/*---------------------------------------------------------------
    local src trg
    src = ext_obj_rel_src_ent_ptr(o)
    trg = ext_obj_rel_trg_ent_ptr(o)
    WRITEF :SOURCE                      :dmr_mem_name(src)
    WRITEF :FORWARD-VERB             ":ext_obj_rel_fwd_verb(o):":
   WRITEF :SOURCE-MANDATORY         :ext_obj_rel_src_mandatory(o)
    WRITEF :SOURCE-MAXIMUM-CARDINALITY
":ext_obj_rel_src_max_card(o):":
   WRITEF :SOURCE-MINIMUM-CARDINALITY  :ext_obj_rel_src_min_card(o)
   WRITEF :SOURCE-MEDIAN-CARDINALITY   :ext_obj_rel_src_med_card(o)
    WRITEF :TARGET                      :dmr_mem_name(trg)
    WRITEF :INVERSE-VERB             ":ext_obj_rel_inv_verb(o):":
   WRITEF :TARGET-MANDATORY         :ext_obj_rel_trg_mandatory(o)
    WRITEF :TARGET-MAXIMUM-CARDINALITY
":ext_obj_rel_trg_max_card(o):":
   WRITEF :TARGET-MINIMUM-CARDINALITY  :ext_obj_rel_trg_min_card(o)
   WRITEF :TARGET-MEDIAN-CARDINALITY   :ext_obj_rel_trg_med_card(o)
/*---------------------------------------------------------------
/* Terminator:
/*---------------------------------------------------------------
    WRITEF;
```

BACHREC

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHREC - Reconcile executive for Bachman Import example:
/*---------------------------------------------------------------
    local o         /* object number
    local name      /* used for name editing
    parse arg o     /* get object number
/*---------------------------------------------------------------
/* Common name editing:
/*---------------------------------------------------------------
    name = UPPER(ext_obj_name(o))
    name = TRANSLAT(name,:--:,: _:)
    name = REDUCE(name,29,:-:)
    name = SUBSTR(name,4)
    name = STRIP(name)
/*---------------------------------------------------------------
/* Check object type and take appropriate action:
/*---------------------------------------------------------------
    if ext_obj_type(o) eq :ENTITYMODEL:  then goto entitymodel
    if ext_obj_type(o) eq :ENTITY:       then goto entity
    if ext_obj_type(o) eq :RELATIONSHIP: then goto relationship
    exit 8                               /* can’t happen!
/*---------------------------------------------------------------
/* Reconciliation for ENTITY MODEL:
/*---------------------------------------------------------------
   -entitymodel
    dmr_mem_name(o)  = :EM-:name
   dmr_mem_type(o)  = :GROUP:  /* not rigorous but convenient... */
    goto exit00
/*---------------------------------------------------------------
/* Reconciliation for ENTITY:
/*---------------------------------------------------------------
   -entity
    dmr_mem_name(o)  = :EN-:name
    dmr_mem_type(o)  = :ENTITY:
    goto exit00
/*---------------------------------------------------------------
/* Reconciliation for RELATIONSHIP:
/*---------------------------------------------------------------
   -relationship
    dmr_mem_name(o)  = :RL-:ext_obj_key(1):-:name
    dmr_mem_type(o)  = :BUSINESS-RELATIONSHIP:
    goto exit00
/*---------------------------------------------------------------
/* Exit:
/*---------------------------------------------------------------
   -exit00
    exit 0
```

BACHPREV

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHPREV - PREVIEW executive for Bachman Import example:
/*---------------------------------------------------------------
    local o          /* object number
    parse arg o      /* get object number
/*---------------------------------------------------------------
/* Check object type and take appropriate action:
/*---------------------------------------------------------------
    if ext_obj_type(o) eq :ENTITYMODEL:  then MPR :BACHPRMOD :o ;
    if ext_obj_type(o) eq :ENTITY:       then MPR :BACHPRENT :o ;
    if ext_obj_type(o) eq :RELATIONSHIP: then MPR :BACHPRREL :o ;
```

BACHPRMOD

```
mpxx literal=:
/*---------------------------------------------------------------
/* BACHPRMOD - PREVIEW executive for Bachman MODEL:
/*---------------------------------------------------------------
    local o         /* object number
    parse arg o     /* get object number
/*---------------------------------------------------------------
/* Repository update function, member name, member type:
/*---------------------------------------------------------------
    WRITEF dmr_mem_func(o) dmr_mem_name(o) ;
    WRITEF dmr_mem_type(o)
/*---------------------------------------------------------------
/* Common clauses (alias type is 'SHORT'):
/*---------------------------------------------------------------
    MPR :MPDYMMLOCC :o: SHORT ;:
/*---------------------------------------------------------------
/* CONTAINS clause:
/* (SEE clause for now):
/*---------------------------------------------------------------
    local c x                /* chained objects
    c = 1                    /* first child
    x = : :                  /* separator
    if TYPE(ext_obj_chain(c)) eq :N: then -
      WRITEF :SEE:
    do while TYPE(ext_obj_chain(c)) eq :N:
      local ch                          /* child object
      ch = ext_obj_chain(c)             /* child number
      WRITEF :  :x°°dmr_mem_name(ch)    /* output child name
      x = :,:                           /* update separator
      c = ext_obj_chain(c)              /* next child in chain
    end
/*---------------------------------------------------------------
/* Terminator:
/*---------------------------------------------------------------
    WRITEF;
```

BACHPRENT

```
mpxx literal=:
/*--------------------------------------------------------------
/* BACHPRENT - PREVIEW executive for Bachman ENTITY:
/*--------------------------------------------------------------
    local o          /* object number
    parse arg o      /* get object number
/*--------------------------------------------------------------
/* Repository update function, member name, member type:
/*--------------------------------------------------------------
    WRITEF dmr_mem_func(o) dmr_mem_name(o) ;
    WRITEF dmr_mem_type(o)
/*--------------------------------------------------------------
/* Common clauses (alias type is 'SHORT'):
/*--------------------------------------------------------------
    MPR :MPDYMMLOCC :o: SHORT ;:
/*--------------------------------------------------------------
/* Entity clauses:
/*--------------------------------------------------------------
    WRITEF :MINIMUM-OCCURRENCE   :ext_obj_ent_minvol(o)
    WRITEF :MAXIMUM-OCCURRENCE   :ext_obj_ent_maxvol(o)
    WRITEF :OCCURRENCE           :ext_obj_ent_expvol(o)
    WRITEF :GROWTH-RATE-PERIOD  ":ext_obj_ent_gptu(o):":
    WRITEF :GROWTH-RATE-PERCENT  :ext_obj_ent_grwpct(o)
/*--------------------------------------------------------------
/* Terminator:
/*--------------------------------------------------------------
    WRITEF;
```

BACHPRREL

```
mpxx literal=:
/*---------------------------------------------------------------
/* Adjust chaining of object (o) to enterprise model (1):
/*---------------------------------------------------------------
     ext_obj_chain(o-1)        = o      /* previous obj to current
      ext_obj_chain_end(1)       = o      /* chain end to current
      ext_obj_parent_pointer(o) = 1      /* current to parent
/*---------------------------------------------------------------
/* Combine partnership (p) data with partnership sets:
/*
/* 1. Get 2 partnership_set keys from partnership (p):
/*---------------------------------------------------------------
      ps_key1 = p_partnership_set(p)   /* key of partnership_set1
      ps_key2 = p_partnership_set2(p)  /* key of partnership_set2
/*---------------------------------------------------------------
/* 2. Convert keys to partnership_set indeces by searching keys:
/*---------------------------------------------------------------
      ps1 = SEARCH(:ps_partnership_setkey:,ps_key1,,,:M:)
      ps2 = SEARCH(:ps_partnership_setkey:,ps_key2,,,:M:)
/*---------------------------------------------------------------
/* 3. Relationship attributes for relationship (o). This is the
/*    core of the model-mapping transform.
/*---------------------------------------------------------------
      local mand1 mand2
                                mand1 = :M:  /* Mandatory
      if p_min1vol(p) eq 0 then mand1 = :O:  /* or Optional
                                mand2 = :M:  /* and again
      if p_min2vol(p) eq 0 then mand2 = :O:
      ext_obj_rel_fwd_verb(o)      = ps_name(ps1)
      ext_obj_rel_inv_verb(o)      = ps_name(ps2)
      ext_obj_rel_src_min_card(o)  = ps_minvol(ps2)  /* invert!
      ext_obj_rel_trg_min_card(o)  = ps_minvol(ps1)
     ext_obj_rel_src_max_card(o)  = ps_maxvol(ps2)  /* here too...
      ext_obj_rel_trg_max_card(o)  = ps_maxvol(ps1)
     ext_obj_rel_src_med_card(o)  = ps_expvol(ps2)  /* here too...
      ext_obj_rel_trg_med_card(o)  = ps_expvol(ps1)
      ext_obj_rel_src_mandatory(o) = mand1
      ext_obj_rel_trg_mandatory(o) = mand2
/*---------------------------------------------------------------
/* Note - if MINIMUM/MEDIAN cardinality is zero, we override with
/*        a value of 1 to conform to ASG model.
/*---------------------------------------------------------------
      if ext_obj_rel_src_min_card(o) eq 0 then -
         ext_obj_rel_src_min_card(o) =  1
      if ext_obj_rel_trg_min_card(o) eq 0 then -
         ext_obj_rel_trg_min_card(o) =  1
      if ext_obj_rel_src_med_card(o) eq 0 then -
         ext_obj_rel_src_med_card(o) =  1
      if ext_obj_rel_trg_med_card(o) eq 0 then -
```

```
         ext_obj_rel_trg_med_card(o) =  1
/*----------------------------------------------------------------
/* Related entities for extracted partnership (p):
/* 1. Get 2 entity keys from partnership_set indeces (ps1,ps2):
/*----------------------------------------------------------------
      e_key1 = ps_entity(ps1)
      e_key2 = ps_entity(ps2)
/*----------------------------------------------------------------
/* 2. Convert keys to entity object indeces by searching for entity
/*    keys in ext_obj_key array:
/*----------------------------------------------------------------
      e1 = SEARCH(:ext_obj_key:,e_key1,,,:M:)
      e2 = SEARCH(:ext_obj_key:,e_key2,,,:M:)
/*----------------------------------------------------------------
/* 3. Entity object indexes for extracted relationship (o):
/*----------------------------------------------------------------
      ext_obj_rel_src_ent_ptr(o) = e1  /* source entity ptr
      ext_obj_rel_trg_ent_ptr(o) = e2  /* target entity ptr
    end
    exit 0
```

The Imported Model

A diagram of the complete entity-relationship model imported is shown in Figure 14 on page 158.

The numbers refer to the keys of the object as defined in the Bachman export file.

Each rounded box represents an entity imported as a member of type ENTITY and each diamond represents a relationship imported as a member of type BUSINESS-RELATIONSHIP.

The complete model is grouped together in the SEE clause of a member of type GROUP for interrogation purposes.

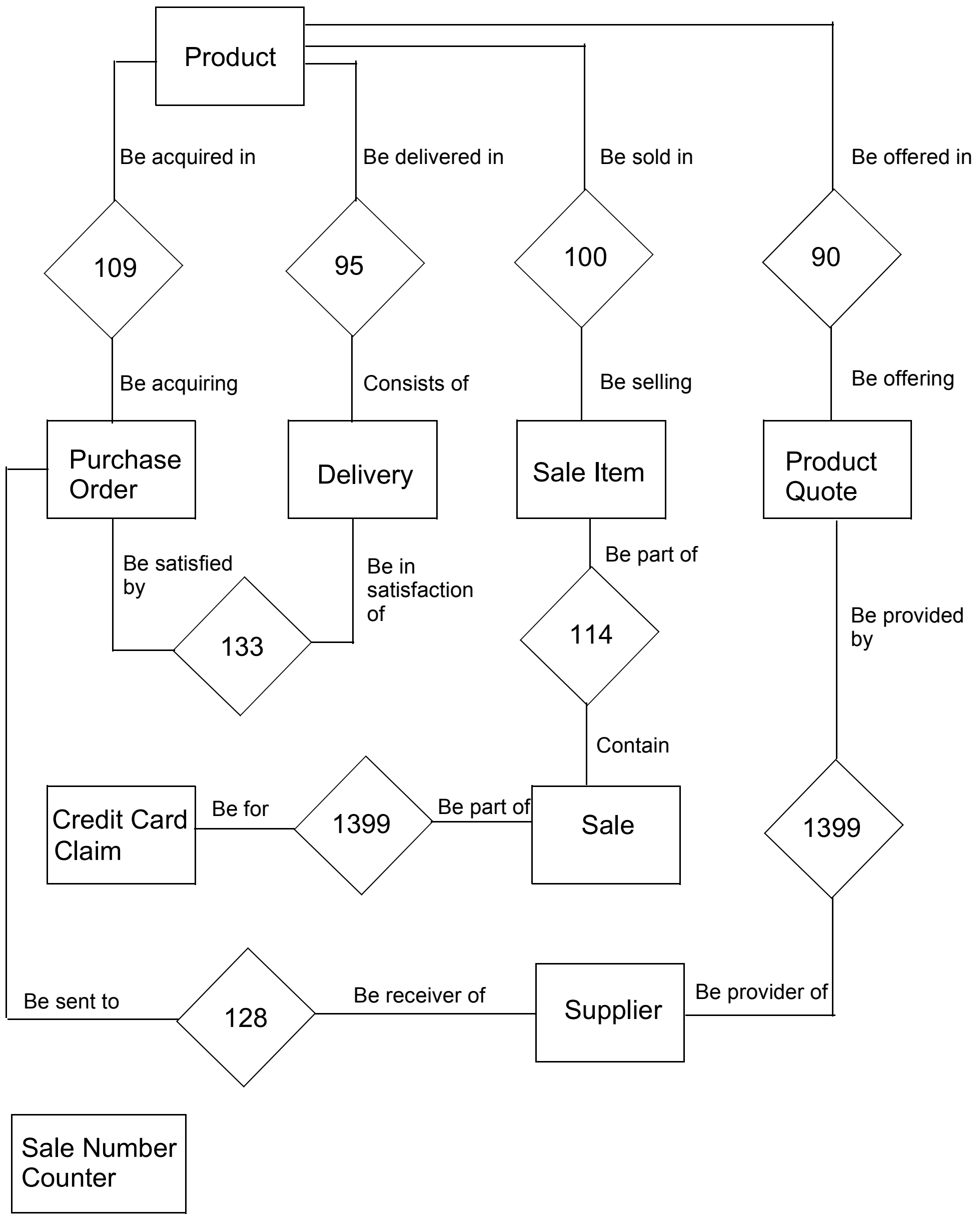


Figure 14. Diagram of the Imported Model

Index